VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce směru pohledu řidiče v obrazech

Driver Eye Gaze Detection in Images

2017 Petr Kolek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Kolek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce směru pohledu řidiče v obrazech
Driver Eye Gaze Detection in Images

Jazyk vypracování: čeština

Zásady pro vypracování:

Inteligentní asisteční systémy se stávají běžným vybavením moderních vozidel. Jedním z takových asistenčních systémů je kamera snímající řidiče, pomocí níž lze detekovat, zda řidič není unavený nebo zda sleduje dění před sebou. Tyto informace pak mohou být využity k varování řidiče v případě náhlé situace, kdy se řidič plně nevěnuje řízení. Cílem této diplomové práce je vytvořit program pro detekci směru pohledu osoby na základě analýzy obrazu získané z kamery citlivé na IR světlo. Tato kamera umožňuje detekci také za zhoršených světelných podmínek.

Ve své práci proveďte:
1. Popište zadaný problém.
2. Analyzujte řešení a popište potřebnou teorii.
3. Proveďte vlastní implementaci řešení a její testování.
4. Zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] Wang, Sung, Venkateswarlu: Eye Gaze Estimation from a Single Image of One Eye, 2003
[2] Chen, Tong, Gray, Ji: A Robust 3D Eye Gaze Tracking System using Noise Reduction, 2008
[3] Hansen, Ji: In the Eye of the Beholder:A Survey of Models for Eyes and Gaze, 2010
[4] Chennamma, Yuan: A Survey on Eye-Gaze Tracking Techniques, 2013

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Michael Holuša**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval svému vedoucímu diplomové práce Ing. Michaelovi Holušovi za jeho cenné rady a čas, který mi věnoval. Dále bych chtěl poděkovat rodině za poskytnutí podpory a zázemí během studia.

Abstrakt

Tato diplomová práce se zabývá problematikou detekce směru pohledu. Jsou v ní uvedeny metody pro sledování a určování směru pohledu. Následně jsou popsány metody pro zpracování obrazu, problémy související s detekcí oka a směru pohledu. Pro samotnou implementaci je zvolena metoda pomocí jedné IR kamery. Hlava je detekována pomocí kaskádových klasifikátorů a landmarků. Detekce středu oka (zorničky) je provedena pomocí algoritmu založeném na orientaci gradientů. Výpočet směru pohledu je poté spočítán jako vychýlení středu zorničky od skutečného středu oka. Algoritmus se osvědčil na testovacích videích s přesností do 4°. Vytvořené řešení umožňuje detekovat směr pohledu z IR kamery a pomocí konfiguračního souboru doladit přesnost detekce. Přínosem této práce je, že jsme schopni detekovat směr pohledu řidiče pouze z kamery.

Klíčová slova: detekce, směr pohledu, OpenCV, detektor, LBP, HOG, HAAR, orientace gradientů, landmark

Abstract

This thesis deals with the detection of the view direction. The methods for monitoring determining the direction of view are includes in the thesis. Next, there are described methods of image processing, the problems associated with the detection of the eye and the direction of view. For the actual implementation is selected the method uses one IR camera. The head is detected using cascade classifiers and landmarks. Detection centre of the eye (pupil) is performed using an algorithm based on the gradient direction. The calculation of the view direction is calculated then as the deflection of the centre of pupil from the real centre of the eye. The algorithm has proved using test videos with an accuracy of up to 4°. The created solution enables to detect the direction of view from the IR camera and tune the detection accuracy using the configuration file. The benefit of this work is that we are able to detect the direction of the driver's view only from the camera.

Key Words: detection, direction of view, OpenCV, LBP, HOG, HAAR, landmark, gradient orientation, detector

Obsah

Seznam použitých zkratek a symbolů

MLL	–	Machine Learning Library
IR	–	Infra-Red
FPS	–	Frame Per Second
EOG	–	Electrooculogram
LBP	–	Local Binary Patterns
HOG	–	Histogram of Oriented Gradients
HAAR	–	HAAR-like features
3D	–	ThreeDimensional
RGB	–	Red Green Blue
XML	–	eXtensible Markup Language
ROI	–	Region Of Interest
API	–	Application Programming Interface

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato diplomová práce se zabývá detekcí směru pohledu osoby na základě analýzy obrazu získaného z kamery citlivé na IR světlo. Sledováním směru pohledu osoby jsme schopni určit, zdali je řidič unaven nebo zda se plně věnuje řízení a sleduje dění před sebou. Tuto možnost nabízejí v dnešní době tzv. inteligentní asistenční systémy řidiče, které můžeme nalézt v moderních vozidlech a které se v nich budou objevovat čím dál častěji. Využitím inteligentních asistenčních systémů řidiče se může předcházet k nehodám, včetně těch, které končí tragicky. Tyto systémy bývají doplněny o další funkčnosti jako například detekcí frekvence mrkání. Podobné systémy lze nalézt i ve zdravotnictví, studiu psychologie člověka, ve vnímání designu, marketingu nebo při interakci člověka a počítače.

Cílem této diplomové práce je vytvořit program pro detekci směru pohledu osoby na základě analýzy obrazu získané z kamery citlivé na IR světlo. Tato kamera umožňuje detekci také za zhoršených světelných podmínek.

Tato práce je rozdělena do několika samostatných kapitol. Na začátku každé kapitoly lze nalézt seznámení se s obsahem kapitoly a je v něm shrnut obsah dané kapitoly.

V druhé kapitole se zabýváme analýzou problému a problematikou týkající se vlastností oka, možnostmi sledování zraku různými metodami a přístroji a následně získanými daty pro jednotlivé aplikace. V závěru druhé kapitoly se nachází výčet různých typů přístrojů sloužících k detekci směru pohledu.

Třetí kapitola je zaměřena na počítačové vidění, obraz, operace s obrazem a možnosti detekce objektů. Je v ní vysvětleno, co je to počítačové vidění a jak je reprezentován obraz. Následuje pak popis knihovny OpenCV, která je využita v praktické části této práce. Vybrané metody, které jsou následně použity v praktické části jsou také obsahem této kapitoly. Jedná se zejména o metody detekce objektů a metody úpravy obrazu.

Čtvrtá kapitola obsahuje implementační část detektoru směru pohledu implementovaného v programovacím jazyce C++ a s použitím knihovny OpenCV. V této kapitole je popsán algoritmus pro detekci středu rohovky a algoritmus pro vyhledání landmarků obličeje. Navazujícím obsahem je výpočet směru pohledu uživatele z dat získaných na základě těchto algoritmů a jejich vylepšení za účelem zisku vyšší přesnosti. V závěru této kapitoly je popsáno urychlení výpočtů detekce směru pohledu s využitím OpenMP a ovládaní programu.

Předposlední kapitola se zabývá zhodnocením výsledků a přesnosti detekce. Obsahuje také ukázku případů pořízených během testování detektoru a grafy znázorňující přesnost detektoru pro danou osu.

V poslední kapitole je závěr, ve kterém je shrnuta celá tato práce.

2 Analýza problému - problematika

Tato kapitola se zabývá analýzou problematiky detekce směru pohledu. Obsahuje také vybrané možnosti detekce oka, středu oka a směru pohledu.

2.1 Oko

Lidské oko je pro člověka důležité pro vnímání okolí a nachází se úplně na začátku obrazového řetězce člověka. Oko je tvořeno dvěma hlavními částmi, kterými jsou rohovka (cornea), což je vnější část, oka a čočkou (lens), která je ukryta uvnitř oka. Tyto dvě části dohromady fungují jako fotoaparát. To kolik světla dopadá do oka, řídí duhovka (iris), která se nachází mezi rohovkou a čočkou. Otvor v duhovce, kterým prostupuje světlo do oka se nazývá zornice (pupil). Po vstupu světla skrz zornici postupuje světlo dále skrz průhledný sklivec (vitreous humor) až na světlocitlivou sítnici (retina), kde vytváří otočený obraz. Na sítnici se nachází velké množství světlocitlivých buněk. Obsahuje kolem 130 miliónů tyčinek a kolem 7 miliónů čípků. Tyčinky jsou velmi citlivé a díky nim můžeme vidět v šeru (jen černobíle), čípky jsou méně citlivé, ale umožňují rozpoznávat barvy. Tímto tedy můžeme říci že oko je velmi citlivý fotoaparát, který se dá přirovnat k CCD snímači nebo kinofilmu. Na sítnici se nachází ještě jedna část oka nazývána žlutá skvrna (fovea). Žlutá skvrna je místem s největším rozlišením a díky ní vidí oko nejostřeji. Obsahuje největší hustotu světločivých orgánů (zde čípků), postupně směrem dále od žluté skvrny čípků ubývá a přibývá tyčinek. Každý člověk může mít žlutou skvrnu umístěnou trochu jinde[4].

Obrázek 1: Schéma oka [4]

Vidění člověka není tedy v celém obrazu stejně ostré. Pokud vnímáme obraz skrze žlutou skvrnu, nazýváme toto vidění fovealní a naopak vnímání obrazu mimo žlutou skvrnu nazýváme periferní vidění. Mezi foveálním a periferním vidění se ještě nachází parafovealní, vidění viz Obr. 2. Periferní vidění má za úkol kontrolovat zajímavá místa a důležité objekty, na které člověk

v nejbližší době zaměří fixací zraku. Periferní vidění má zhoršenou ostrost oproti foveálnímu vidění zhruba o 15 - 50%. Tento jev ale člověk běžně nezaznamená, jelikož oko je stále v pohybu. Přesouvá se skokově okolo cca. každých 50ms a zrakové vnímání je v této době vypnuto. Těmto přechodům se říká sakády. Naopak při zastavení oka na dobu delší než 0,1s se říká fixace. Sakády tedy zprostředkovávají pomocí periferního vidění přenos fixačních bodů z místa A do místa B podle vybraného cíle mozkem.

Obrázek 2: Ostrosti vidění [4]

2.2 Získaná data a jejich reprezentace

Po zpracování dat mnohdy potřebujeme nějakým způsobem zobrazit naměřené hodnoty v obraze a ne si pouze hodnoty uložit do nějaké datové struktury. Toho lze docílit i přímou úpravou obrazu. Pro reprezentaci směru pohledu můžeme do obrazu zakreslit šipku reprezentující směr pohledu (animovaného zobrazení). Pro marketing se používají tzv. Heat Mapy[13] na kterých můžeme vidět kam se lidé dívají nejčastěji. Níže jsou vypsány některé typy reprezentace těchto dat.

1. **Animované zobrazení** - tato reprezentace se používá nejčastěji pokud potřebujeme vědět polohu bodu v každém snímku zvlášť. K této metodě můžeme implementovat i zobrazení čáry kde se daný bod zpětně nacházel (třeba 10 snímků zpětně).

2. **Statické zobrazení** - je podobné animovanému, ale s tou vyjímkou, že body zůstanou v obraze. Můžeme si například ukládat pro každý snímek, na jaké pozici se hledaný bod nacházel.

3. **Heat-mapa** - patří mezi jednu z nejvyužívanějších reprezentací dat získaných detekcí směru pohledu a představuje jednu ze statistických metod určených pro skupiny uživatelů. Zjišťuje, kde se uživatelé dívají nejčastěji. Využití těchto map můžeme vidět v marketingu (kde se má zvolit jaká reklama nebo zda-li reklama uživatele něčím zaujala).

4. **Blind zones maps** - Slepé zóny fungují na opačném principu jako heat-mapy. Vizualizují se méně prohlížené oblasti a tím pádem nám umožňují snadnější pochopení relevantních informací.

2.3 Možnosti sledování zraku

V této části se pokusím popsat různé metody sledování zraku. Mezi možné způsoby sledování směru pohybu oka lze zařadit metody na principu elektrostatického sledování oka, mechanického sledování oka nebo optického sledování oka, které můžeme dále rozdělit na kontaktní a bezkontaktní. Když se zaměříme na technologický pokrok, tak si určitě můžeme všimnout, že zařízení pro detekci směru pohledu jsou pro uživatele příjemnější než kdysi. Všechny části se daří dále zmenšovat včetně senzorů, ale stále nikdo nebyl schopen vyrobit ideální zařízení. Aby bylo zařízení ideálním, mělo by splňovat alespoň tyto vlastnosti:

- Zpracování v reálném čase
- Snímání provádět bez jakýchkoliv kontaktních měřidel
- Snímání by nemělo být ovlivněno okolním prostředím (šero a stíny)
- Uživatel by neměl být omezený ve výhledu
- Binokulární snímání, které nám umožní vidět oběma očima, kdy se nám obrazy spojí v jeden a díky tomu můžeme vnímat hloubku
- Snímání by mělo fungovat na všech osobách
- Chybovost pod 1%

2.3.1 Mechanické snímání

Mechanické snímání je nejstarší a jeho používání nebylo pro uživatele nejpříjemnější. Většinou k oku bylo připojené mechanicky nějaké zařízení, které snímalo pohyb oka [6]. V roce 1879 popsal pan Javal jak se chová lidské oko při čtení. Pozoroval totiž, že čtenářův pohyb očí není při čtení plynulý ale přeskakuje k další fixaci. Jeho metoda byla založena jen na pozorování pouhým pohledem na zrcátko, které bylo umístěné před čtenářem. Další následník v roce 1898 byl Delabarre, který k oku připevnil plošku s vláknem spojenou s táhlem a ta zaznamenávala horizontální pohyby oka na sklíčko. Ovšem v šedesátých letech minulého století byla jeho metoda vylepšena tím, že se využívalo podtlaku, kterým se na oko připevnila čočka. Na tuto čočku se dále připevnilo zrcátko, které sledovalo směr odraženého světla daleko lépe než při obyčejné pozorování rohovky. Další metoda obsahovala v čočce cívku a směr pohybu oka byl měřen jako změna napětí v elektromagnetickém poli kolem oka.

Tyto metody se již v dnešní době nepoužívají i přes to, že byly velice přesné, ale zároveň pro pozorovaného nepříjemné.

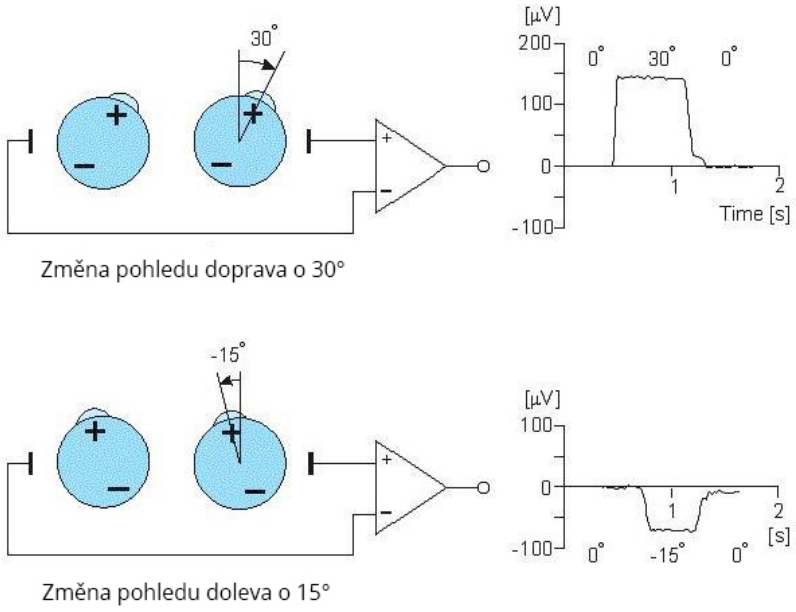
2.3.2 Elektrostatické snímání

Další metodou měření směru pohledu může být také elektrostatické snímání [10][12]. Toto snímání využívá elektrod připojených k okolí oka. Oči je možno snímat i v absolutní tmě a dokonce

i když má uživatel oči zavřené. Výsledný signál, který dostaneme z elektrod umístěných kolem oka se nazývá Electrooculogram(EOG) 3. Pokud se rohovka přesune ze středové pozice k libovolnému kraji, tak nastává změna elektrického potenciálu mezi elektrodami. Díky této změně potenciálů mohou být oči sledovány.

Nicméně kvůli tomu, že EOG signál pouze udává změnu pohybu a neudává přesnou pozici toho kde je oko umístěné tak je velmi obtížné používat EOG k detekci pomalých pohybů oka a k detekci směru pohledu. Nevýhodou tohoto systému tedy je, že ve srovnání se systémy založenými na principu videa je jeho přesnost nižší.

Firma NNT DoCoMo vyvinula systém, který umožňuje ovládat např. audio přehrávač právě pomocí pohybu očí. Zařízení vypadá jako obyčejná sluchátka, ale elektrody jsou obsaženy v samotných sluchátkách. Pro uživatele je pak nošení takovéhoto zařízení praktičtější než nalepené elektrody na obličeji.



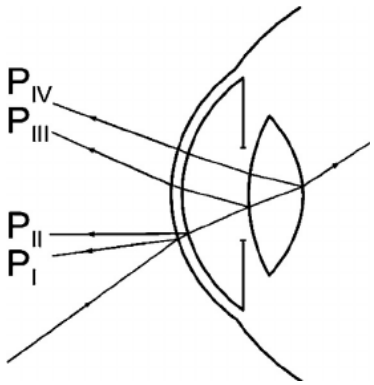
Obrázek 3: Ukázka změny pohledu a EOG signálu. [9]

2.3.3 Optické snímání

Optické snímání (foto-okulografie a video-okulografie) patří mezi nejrozšířenější techniku pro sledování očních pohybů. Výhodou je, že pro implementaci nám stačí pouze použít fotoaparát nebo videokameru. Díky tomu je cena na výrobu takovéhoto zařízení nižší než u jiných metod. Bohužel tato metoda je dosti závislá na kvalitě obrazu. Obraz by měl mít velmi dobrý kontrast, aby bylo možné najít střed duhovky. Základem je nutné použít dobrou kameru, která bude umět pořizovat videozáznam i za zhoršených podmínek např. IR kameru. Duhovka bývá často částečně zakrytá očním víčkem a poté se nám ztěžuje měření vertikálních pohybů oka. Avšak někdy nastává další problém, kterým je malý kontrast mezi duhovkou a zorničkou. Můžeme

dosáhnout i opačného efektu a tím je naopak rozsvícení zorničky. Toho dosáhneme pomocí správně umístěného světla. Zdroj světla bude rovnoběžný spolu s optickou osou kamery a světlo které dopadne na zadní stěnu oka se odrazí zpět ke kameře a tím se nám zornička rozsvítí. Na stejné principu funguje i efekt červených očí při fotografování. Na takový obraz můžeme použít metodu prahování pro zjištění polohy zorničky.

Pokud použijeme IR kameru, tak můžeme v oku spatřit světlé odlesky způsobené právě IR přísvitem tzv. Purkyňovy obrazy, viz Obr. 4. Velké množství popsaných metod využívají právě těchto odlesků k detekci směru pohledu. Při dopadu světla do oka se část odráží zpět a tím vznikají ony odlesky [2]. Purkyně zdokumentoval celkem 4 druhy odrazů. První se nachází v místě rohovky a vzduchu. Vzhledem k tomu, že index lomu mezi rohovkou a vzduchem je velký, tak je i intenzita tohoto odlesku největší. Druhý se nachází na pomezí rohovky a oční komory a je méně intenzivní. Druhý odlesk často splývá s prvním. Třetí je tvořen na rozhraní oční komory a čočky. Čtvrtý odlesk vzniká na rozhraní čočky a sklivce. Ovšem metody detekce směru pohledu založených na těchto odrazech využívají první dva odlesky, protože třetí a čtvrtý je obtížné zaznamenat. Tyto odlesky jsme schopni následně použít pro detekci směru pohledu a to tak, že spočteme směrový vektor ze středu zorničky a nalezeného odlesku. Tato metoda většinou vyžaduje kalibraci pro každého uživatele.



Obrázek 4: Purkyňovy obrazy (odrazy)

K optickému snímání můžeme využít různý počet kamer z různých úhlů a libovolný počet světel. Ovšem abychom mohli zaznamenat rychlejší pohyby např. chvění oka, potřebovali bychom vysokorychlostní kamery, které jsou schopny zaznamenat více snímků za vteřinu (FPS). Systémy založené na optickém snímání můžeme dále rozdělit do dalších kategorií. Rozdělení může být na náhlavní systémy a bezkontaktní systémy. Systémy mohou využívat pouze přírodního osvětlení nebo naopak různé druhy přísvitu (IR).

2.3.4 Náhlavní systémy

Téměř vždy se do této kategorie řadí různé druhy brýlí, helem nebo jiných zařízení, která jsou připevněna k hlavě a jsou vybavena nějakou kamerou. Nevýhodou tohoto řešení je, že uživatel

musí mít na hlavě dané zařízení, které ho může omezovat v dalších aktivitách.

Ing. Marcela Fejtová spolu se svým bratrem vyvinuli na katedře kybernetiky ČVUT v Praze systém nazvaný **I4Control** [16]. Základem je malá kamera, která je přichycena k obrubě obyčejných brýlí. Kamera sleduje uživatelovy oční pohyby. Sledování směru pohledu se provádí pomocí detekce zorničky. Brýle se připojují k počítači pomocí USB portu. Obsluha zařízení je velmi jednoduchá. Uživatel si nasadí brýle a podle toho kam se dívá se pohybuje i myš po obrazovce. Klikání na myš je pak ovládáno zavřením oka na nastavenou dobu. Využití těchto brýlí je zejména pro tělesně a psychicky handicapované uživatele.

Obrázek 5: Brýle I4Control

Na podobném principu fungují brýle **Tobii Pro Glasses 2** od stejnomenné společnosti Tobii [8]. Brýle Pro Glasses 2 jsou jiné v tom, že neslouží pro přímé ovládání počítače, ale slouží pro pochopení jak se lidé chovají ve společnosti. Tyto brýle zaznamenávají video kam se člověk díval a co konkrétně sledoval. Využití lze nalézt i v automobilu, kdy prostřednictvím těchto brýlí jsme schopni zjistit např. zda uživatel sledoval dopravní značky a zda se plně věnoval řízení, nebo při studiu chování člověka (co ho zaujalo).

Obrázek 6: Brýle Tobii Pro Glasses 2

Piloti stíhacích letounů mají ve svých helmách zařízení [14], umožňující zaměření jejich cíle. Proto aby pilot zaměřil vybraný cíl musí provést natočení hlavy směrem k cíli. Měření je založeno na změně magnetického pole uvnitř kokpitu při natočení hlavy. Ovšem zjišťovat směr pohybu přímo z helmy je složitější a méně přesné, proto se stále zůstává u měření magnetického pole.

Obrázek 7: Helma pro stíhací piloty model JHMCS [15]

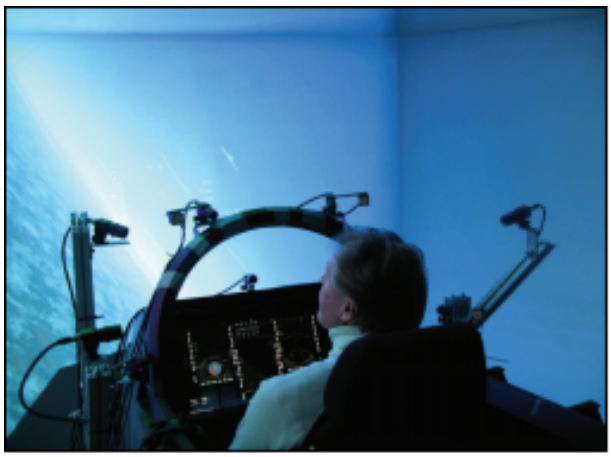
2.3.5 Bezkontaktní systémy

Systémy tohoto typu nepotřebují žádná zařízení, která by byla připevněna na hlavu uživatele. Většinou jsou to různé kamery, které snímají uživatele. Zařízení tohoto typu mohou být složena z různých druhů i počtu kamer. Některé bezkontaktní systémy počítají s natočením hlavy, jsou to zejména ty, které obsahují více kamer a další, které využívají pouze jedné kamery. Na základě detekce směru pohledu z kamery umístěné před řidičem můžeme rozlišit zóny kam se řidič v automobilu dívá Obr. 8 (1. řidičovo okénko, 2. řidičovo zpětné zrcátko, 3. řidičovo stínítko, 4. spolujezdcovo stínítko, 5. spolujezdcovo okénko, 6. spolujezdcovo zpětné zrcátko, 7. přihrádka spolujezdce, 8. středový panel, 9. volant, 10 - 18 pohle skrz čelní sklo rozdělený do zón.

Obrázek 8: Čelní výhled z automobilu a rozdělení do zón

Zjistit ve stíhacím letounu kam se pilot dívá je velmi důležité k tomu, aby mohl zaměřit na vybraný cíl. Článek [14] popisuje řešení používající kamery uvnitř kokpitu, kdy se detekce směru pohledu zjišťovala právě pomocí těchto kamer viz Obr. 9. Kamery byly rozmístěny zhruba

v odstupech 30° a střídavě nahoře a dole okolo sedačky. Z naměřených dat se dále mohl vypočítat 3D směrový vektor pohledu. Přesnost měření byla v rozmezí 1-3°.



Obrázek 9: 8 IR kamer pro detekci směru pohledu

Tobii Pro Spectrum je bezkontaktní systém pro sledování směru pohledu. Před uživatelem je položen panel obsahující 2 kamery, kde každá z kamer zaznamenává video s 600 FPS. Můžeme si zakoupit buď samostatný panel nebo set panelu s obrazovkou. Pro Spectrum umožňuje detekovat směru pohledu včetně natočení hlavy do úhlu 30° a zaznamenat i velmi rychlé pohyby. Pro každého uživatele se musí provést kalibrace. Výhodou tohoto zařízení je to, že dokáže zpracovávat data s krátkou dobou pozornosti např. u kojenců a dětí, které se nedokáží delší dobu soustředit. Systém je velmi robustní, díky čemu patří mezi jeden z nejoblíbenějších systémů.

Obrázek 10: Set Tobii Pro Spectrum

3 Počítačové vidění, obraz a operace s obrazem

Tato kapitola popisuje knihovnu OpenCV, jak se snímá obraz, jak je reprezentován a zpracováván. Jsou zde popsány obrazové funkce, které byly použity následně v praktické části této práce, zabývající se implementací samotného detektoru směru pohledu.

3.1 OpenCV

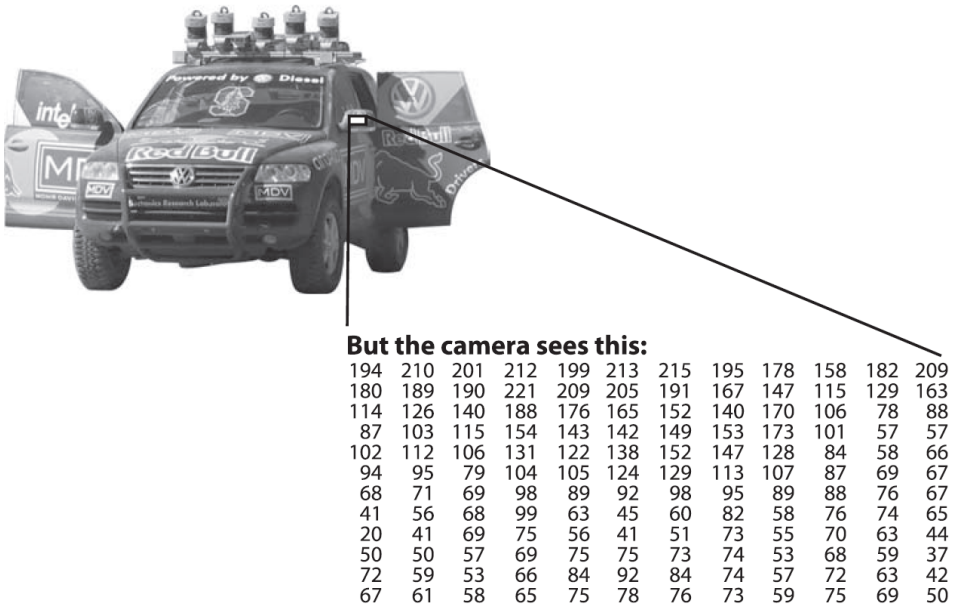
OpenCV je opensource knihovna napsána v jazyce C++, sloužící ke zpracování obrazu [1]. Knihovna může být využita společně se spoustou jiných programovacích jazyků (např. Python a Matlab) a operačních systémů. Tato knihovna obsahuje stovky funkcí, které můžeme využít pro širokou škálu sofistikovaných řešení od zpracování obrazu ve strojovém vidění, medicíně, zabezpečení, uživatelském rozhraní, kalibraci kamer, zpracování obrazu z více kamer až po robotiku. Vzhledem k tomu, že v dnešní době jde počítačové vidění dopředu souběžně se strojovým učením, tak OpenCV obsahuje plnou podporu pro strojové učení (MLL). Tato MLL část knihovny je velmi výkonná a vhodná pro úlohy řešící zpracování obrazu, ale může být využita i v jiných úlohách. Je zaměřena na statistické rozpoznání vzorů a k jejich kategorizaci. V roce 2014 byla vydána verze OpenCV 2.4, která umožňuje zpracovávat obraz i na mobilních zařízeních s operačním systémem Android. Další významná verze byla velmi optimalizována verze OpenCV 3. Výhodami knihovny je tedy velké množství vytvořených funkcí a kvalitní dokumentace. Pro aplikace pracující v reálném čase nejsou vždy všechny její funkce vhodné, jelikož jsou pomalejší, než ty, které si může vývojář implementovat sám.

Kdo OpenCV používá? Knihovna je využívána samostatnými programátory, studenty i velkými nadnárodními společnostmi, mezi které můžeme zařadit např. Google s jejich Street View mapami, kde využívají techniky pro kalibraci obrazu a jeho spojování. Mezi jiné velké hráče na trhu využívající knihovnu OpenCV můžeme zařadit Intel, Yahoo, Microsoft, IBM, Sony, Honda, Toyota nebo i některé nové, ačkoliv malé startupy. Aplikace využívající tuto knihovnu můžeme takto nalézet po celém světě, kde kontrolují výrobní linky, správnosti štítků na obalech produktů, detekci narušitele vymezeného prostoru, runwaye mezinárodních letišť, navigaci robotů nebo právě detekci únavy řidiče.

3.2 Digitální obraz

Obraz je reprezentace dat pořízených např. z fotoaparátu nebo videokamery, se kterou je následně možné provádět různé operace a vizualizovat je. Obvykle bývá reprezentován pomocí čísel (binární nebo hexadecimální soustavy). Existují dva typy obrazu a to rastrový a vektorový. Rastrový obraz můžeme pořídit právě pomocí videokamery a výsledný obraz je popsán pomocí samotných obrazových bodů tzv. pixelů. Pixely jsou umístěny v pravidelné mřížce, kde každý má svou polohu a hodnotu. Pixel může obsahovat hodnoty RGB - třísložková hodnota s využitím pro barevný obraz, nebo jednosložkový pro černobílý obraz. Naopak vektorový obraz spoléhá

na matematické výpočty a vizualizuje se pomocí čar a křivek. Tyto obrázky se vytváří ve speciálních programech určených k tvorbě vektorové grafiky. Na obrázku 11 níže je tedy vidět co vidí člověk a co vidí počítač (rastrový obraz). Pro získání plynulého videa potřebujeme abychom zaznamenali, alespoň 25 takových obrazů za sebou během jedné vteřiny.



Obrázek 11: Počítač 'vidí' výřez zpětného zrcátka pouze jako mřížku reprezentovanou hodnotami intenzity pixelu [1]

Jelikož lidé vnímají okolí zejména pomocí očí, tak se může stát, že si někteří mohou myslet, že počítačové vidění je jednoduché tak jako lidské. To ovšem není pravdou, jelikož ani lidské vidění není jednoduché. U člověka oči odešlou informaci o tom co vidí směrem k mozku a ten na základě svých celoživotních zkušeností odpoví co na obraze je. Člověk např. tedy ví, jak vypadá auto a pokud se mu v mozku spojí vzor v obraze s uloženým vzorem auta, tak dokáže říci, že na obraze je auto. Ovšem ve strojovém vidění nastává problém s tím, že kamera si nedokáže říci tak snadno jako člověk 'vidím rozmazaně - musím zaostřit' nebo 'že si mozek řekne, obraz je moc přesvětlený musím přes duhovku upravit clonu, aby dopadalo méně světla na sítnici'. Tento problém vzniká tím, že člověk má velkou zpětnou vazbu od mozku na svaly, které ovládají oči a dokáží oko 'naladit' do správné konfigurace na rozdíl od kamery.

V počítačovém vidění tedy záleží na tom zda jsme rozpoznali náš hledaný objekt s nějakým objektem v obraze mu podobným. Pro některé oblasti počítačového vidění jsou některé úlohy pro detekci velmi složité a jejich vývoj trvá roky.

3.3 Detekce objektů

Tato část se zabývá detekcí obrazu, to znamená, že v obraze musíme najít oblasti, které jsou pro nás podstatné. V této práci potřebujeme najít obličej osoby, ve kterém se dále budou hledat oči a

význačné tvary obličeje. K tomu, abychom nalezli obličej, budeme potřebovat něco, co vyhodnotí, zda část obrazu obsahuje obličej či nikoli. Právě nyní můžeme využít knihovnu OpenCV, ve které jsou obsaženy klasifikátory hledající podobné objekty. Klasifikátor tedy slouží k určení zda se hledaný vzorek nachází v obraze na základě svého nastavení. Pro to, abychom mohli používat klasifikátory, tak si je budeme muset nejdříve natrénovat. K trénování a rozpoznávání slouží poté algoritmy LBP, HAAR a HOG. O těchto klasifikátorech a jejich provedení se budeme zabývat níže. Každý z klasifikátorů je algoritmicky založen na něčem jiném. To znamená, že časová složitost k natrénování a k rozpoznání obličeje bude pro každý klasifikátor různá. Pro natrénování potřebujeme trénovací množinu, která obsahuje pozitivní a negativní vzorky. Mezi pozitivními vzorky budou obličeje v různých prostředích, velikostech nebo osvětlení. V množině negativních vzorků bude cokoliv jiného kromě obličejů. Pokud si nechceme vytvářet vlastní trénovací data, můžeme použít volně dostupná data z internetu, které ovšem nejsou vytvořena pro všechny druhy objektů. Zpravidla, na čím více vzorcích je klasifikátor natrénovaný, tak je přesnější, ovšem trénováním na vysokém počtu vzorů může dojít k jeho přetrénování. Po natrénování se nám vytvoří soubor ve formátu XML, který se bude poté používat pro rozpoznávání. V dokumentaci OpenCV [3] je uveden přesný návod k trénování.

Detekce objektu se tedy provádí v každém snímku po předzpracování obrazu. Vyzkoušel jsem všechny tři klasifikátory, ale v praxi se nejvíce osvědčil HAAR (byl nejpřesnější), proto byl v praktické části zvolen pro implementaci detektoru směru pohledu.

3.4 Klasifikátory

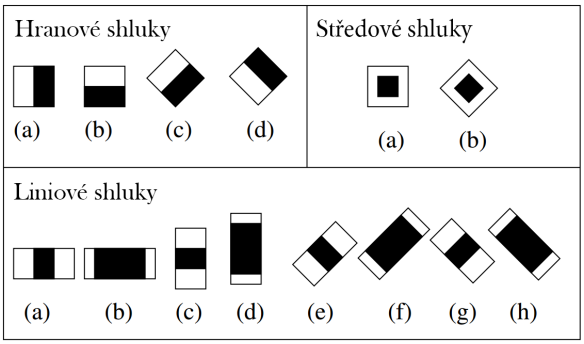
Práce s klasifikátory je po natrénování jednoduchá. Nejdříve si musíme načíst klasifikátor do objektu *CascadeClassifier*, který následně můžeme použít v metodě *detectMultiScale*. Metoda obsahuje vstupy jako je vstupní obrázek, seznam obdélníků, ve kterém budou uloženy nalezené objekty, krok o kolik se má zvětšovat porovnávaný vzor, minimální počet potvrzených vzorků, aby mohla být oblast potvrzena za hledaný objekt, minimální a maximální velikost hledaného objektu. Fungují tak, že začínají na počáteční velikosti okna zadaného uživatelem a postupně metodou 'sliding window' procházejí obrazem. V momentě, kdy dojdou na konec obrazu, zvětší se velikost okna o uživatelem zadanou míru a celý obraz se prochází znovu.

Na Obr. 12 můžeme vidět detekci obličeje v obraze s využitím klasifikátoru. Samozřejmě můžeme v obdélníku s obličejem hledat stejným způsobem oči.

Obrázek 12: Detekce obličeje pomocí HAAR klasifikátoru

3.4.1 HAAR - Haarovy příznaky

Metoda popsána autory Viola a Jones [18]. Haarova metoda je velmi silná a funguje na principu porovnávání dvou sousedních shluků pixelů, přičemž shluky mohou mít různé tvary, natočení a kombinace. Pro detekci světlých a tmavých míst se využívá právě těchto shluků. Na obrázku 13 můžeme vidět základní tvary shluků a jejich aplikaci na obr. 14. Krom těchto základních existují i různé variace těchto shluků.

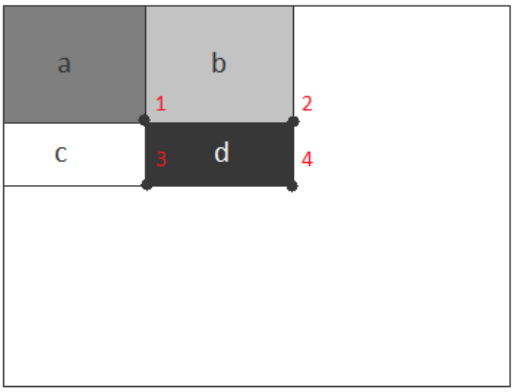


Obrázek 13: Typy HAAR shluků[18]

Obrázek 14: Aplikace vybraného shluku v obraze

Pro hledání příznaků se nastavuje nejmenší možný rozměr příznaku, který se plynule horizontálně nebo vertikálně posouvá obrazem po 1 pixelu. Spočítat sumu hodnot pixelů v buňkách shluků bylo časově velmi složité, proto se přišlo s metodou integrálního obrazu. Ovšem musí se spočíst tento integrální obraz, ale toho lze docílit jediným průchodem přes celý obrázek na začátku zpracování. Integrální obraz velmi urychluje spočtení sumy obsahu vybraného obdélníka a je reprezentován v každém pixelu jako součet hodnot pixelů nad a vlevo od počítaného pixelu.

Poté, pokud chceme vědět hodnotu ve vybraném obdélníku, si vystačíme pouze s přístupem do matice integrálního obrazu, součtem a rozdílem viz Obr. 15.



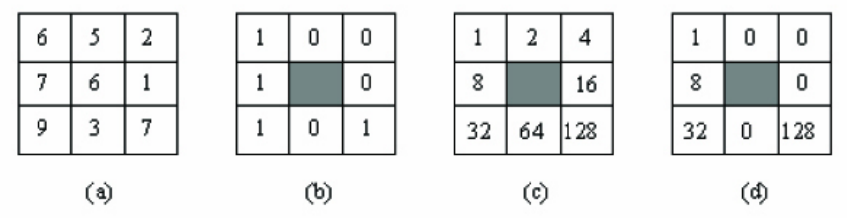
Obrázek 15: Integrální obraz a výpočet[19] - pokud chceme spočíst hodnotu obdélníka d = P(4) + P(1) - P(2) - P(3)

Při hledání shodného vzoru se postupuje kaskádovitě tzn. prochází se obraz v několika iteracích a postupně se přidávají další slabé příznaky pro hledaný objekt. V každé iteraci se algoritmus snaží vyloučit alespoň 50% falešných detekcí.

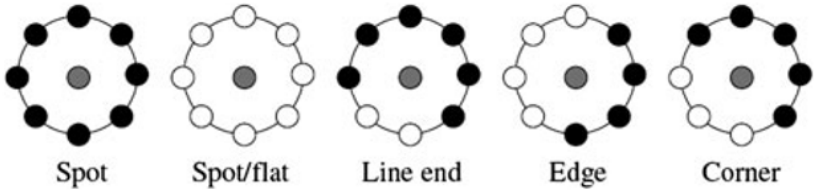
3.4.2 LBP - Lokální binární vzor

Metoda lokálních binárních vzorů (LBP) patří mezi algoritmy pro rozpoznání objektů v obraze. LBP je silná metoda patřící mezi jednu z nejrychlejších. Výhodou této metody je to, že není tolik závislá na úrovni osvětlení obrazu a na rotaci příznaku (hledaného objektu) v obraze. Dokáže detekovat např. hrany, body, plochy a rohy viz Obr. 16. Konfigurací LBP vzoru si tedy můžeme vybrat, jaký tvar budeme detekovat (např. hranu).

Obraz je vhodné před aplikací této metody vhodně upravit tzv. předzpracováním. Doporučuje se provést na obraze *gamma korekce*. Funkce metody, viz Obr. 17, je jednoduchá pro implementaci a funguje následovně. V metodě LBP se pracuje většinou s bloky o velikosti 3x3 pixely, kde se porovnávají okolní pixely vůči středovému pixelu. Hodnota jasu pixelu ve středu bloku je vybrána jako práh pro prahování s okolními pixely. Prahování funguje pouze jako porovnání dvou hodnot a zapsání výsledku. Pokud je tedy hodnota porovnávaného pixelu větší než prahovací hodnota středového pixelu, tak je na místo porovnávaného pixelu zapsána jednička, a naopak. Výsledkem je binární vektor, který se převede do decimálního vektoru pomocí násobící matice viz Obr. 16. Dále se vypočte suma okolních pixelů a výsledné LBP číslo se zapíše do histogramu. V histogramu je uložena informace o počtu výskytů pixelů s danou intenzitou jasu. Tato operace se provede pro všechny pixely v okolí středového pixelu a pro všechny pixely v obraze.



Obrázek 16: Výpočet LBP kódu. (a) Vstupní matice, (b) matice po prahování, (c) násobící matice, (d) matice po vynásobení binárního vektoru s násobící maticí, na závěr se provede součet hodnot z matice (d) a ten se uloží jako středová hodnota [21]



Obrázek 17: LBP - příklady rozpoznávaných tvarů [20]

Porovnáním histogramů tedy zjistíme, jaká je podobnost hledaného vzoru s obrazem v %. Histogramy můžeme porovnat pomocí vzorce viz níže.

$$p = \frac{\sum_{i=0}^{255} min(a_i, b_i)}{\sum_{i=0}^{255} max(a_i, b_i)} \tag{1}$$

Během vývoje v oblasti zpracování obrazů se i tato metoda vylepšuje a vznikají různé variace (např. může se pracovat s maticemi o blocích 9x9).

3.4.3 HOG - Histogramy orientovaných gradientů

Histogramy orientovaných gradientů (HOG) je technika popisující výskyty orientací gradientů v jednotlivých částech obrazů [22]. Základní myšlenka vychází z toho, že všechny objekty mohou být popsány histogramem gradientů. Metoda se skládá z více na sebe navazujících částí.

Předzpracování obrazu - Provádí se z důvodu možného obsahu nežádoucích vlivů v obraze. Vhodné je provést normalizaci jasu a kontrastu. Je možno použít tříkanálový obraz nebo i jednokanálový obraz, tedy obraz ve stupních šedi. Pro další zvýšení robustnosti se doporučuje použít gamma korekci nebo bilaterálního filtru.

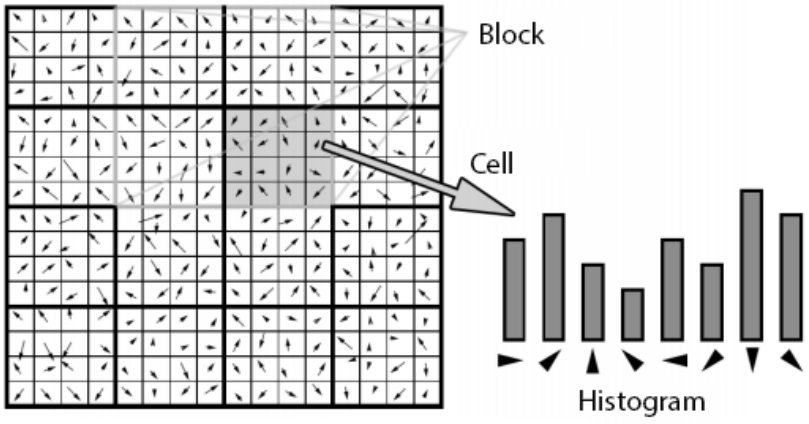
Výpočet gradientů - Gradienty jsou vypočteny na základě vstupního obrazu a masky (nejčastěji 1-D a to ve tvaru [-1,0,1] a $[-1, 0, 1]^T$). Gradienty se vypočtou jak pro osu x, tak pro osu y s transponovanou maskou. Získáme tedy obrazy I_x a I_y.

Histogram orientací gradientu - Po získání gradientů I_x a I_y je pro každý pixel spočtena velikost gradientu m(x,y). Stejně tak je i spočten směr Θ (x,y).

$$m(x,y) = \sqrt{I_x^2 + I_y^2} \tag{2}$$

$$\Theta(x,y) = tan^{-1}\left(\frac{I_y}{I_x}\right) \tag{3}$$

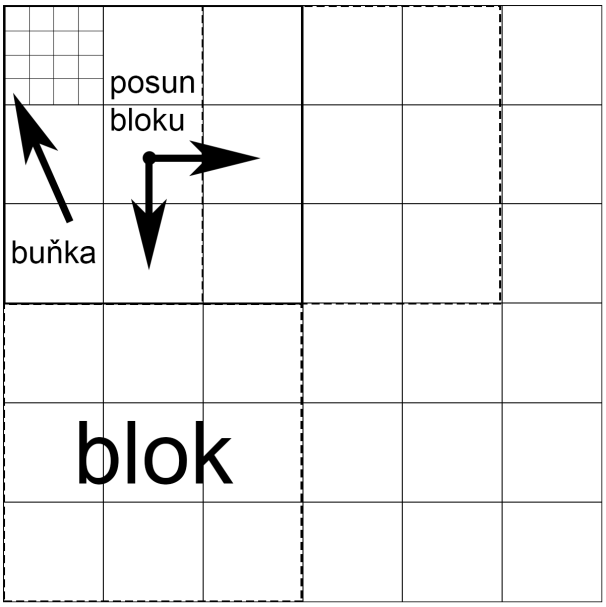
Po výpočtu směru a velikosti gradientu je vypočítán histogram orientací gradientů. Počet binů histogramu je dán rozsahem i a s je počet sektorů. Interval i určuje rozsah v jaké je zjišťován směr gradientu (vetšinou se jedná o interval $< 0; 360° >$ tedy $< 0; 2\pi >$. Číslo s značí, na kolik částí bude rozdělen interval i. Velikosti gradientů jsou přičítány do příslušných binů histogramu a jsou rozděleny i mezi sousední biny, díky čemuž narůstá robustnost konečného deskriptoru viz Obr. 18, který obsahuje 8 sektorů a interval 360°.



Obrázek 18: Histogram orientovaných gradientů vytvořený pro každou buňku [23]

Normalizace histogramu - K tomu, aby byl deskriptor invariantní vůči změně jasu nebo kontrastu, je nutné provést normalizaci histogramu orientací.

Vytvoření deskriptoru - Deskriptor je vytvořen na základě vstupního obrazu, kdy se vstupní obraz rozdělí na čtvercové bloky, viz Obr. 19 (3x3). Bloky se dále rozdělují na buňky (4x4) a pro ně se vypočítají normované histogramy orientací gradientů. Průměrování histogramů buňek uvnitř bloku v jeden získáme popis tohoto bloku. Bloky se při výpočtu překrývají a posunují o zvolený počet pixelů viz Obr.19 Konečný deskriptor vznikne spojení všech popisů samostatných bloků ve vstupním obraze. V případě použití čtvercových bloků je finální deskriptor popisován jako R-HOG. Ovšem můžeme se setkat i s obrazem členěným do kruhových bloků, které se označují jako C-HOG.



Obrázek 19: Schéma buněk a bloků s posunem detekčního okna

3.5 Vyrovnání histogramu

Histogram [3] je grafická reprezentace rozložení intenzit jasu ve vstupním obraze 20 . Vyrovnání histogramu je globální jasová transformace tzn. že hodnota každého pixelu je upravena na základě analýzy celého vstupního obrazu. Nejčastěji se používá, pokud je obraz příliš světlý nebo tmavý. Tyto stavy jsou způsobeny tím, že jednotlivé jasové intenzity jsou příliš blízko sebe. Jde tedy o mapování jednoho rozložení na druhé. Vyrovnáním (ekvalizací) tohoto histogramu dostaneme obraz, který rozloží intenzity jasů lépe (využije větší část histogramu). Po aplikaci této metody můžeme získat v obraze větší kontrast a v obraze nalezneme zřetelnější přechody.

Výhodou této aplikace je i to, že pokud známe funkci ekvalizace, tak jsme schopni zpětně obraz obnovit. Dalšími výhodami je rychlost a jednoduchost implementace. Nevýhodou ovšem může být to, že pokud je v pozadí šum, tak se zvýrazní po aplikaci této metody právě šum a utlumí užitečný signál. Vyrovnání histogramu může být použito u barevných obrazů i v obrazech ve stupních šedi.

Tato funkce je rovněž obsažena v knihovně OpenCV, ale její výpočet není nikterak složitý. Pokud máme obrázek ve stupních šedi tak n_i je celkový počet úrovní šedi i v obraze. Pak pravděpodobnost výskytu úrovně šedé i je dána jako:

$$p(x_i) = \frac{n_i}{n}, i \in 0, ..., L-1, \tag{4}$$

kde L značí celkový počet výskytů šedi v obraze, n celkový počet bodů v obraze. Pak p je diagram normalizovaný do intervalu $[0, 1]$. Poté si definujeme kumulativní distributivní funkci c

odpovídající p:

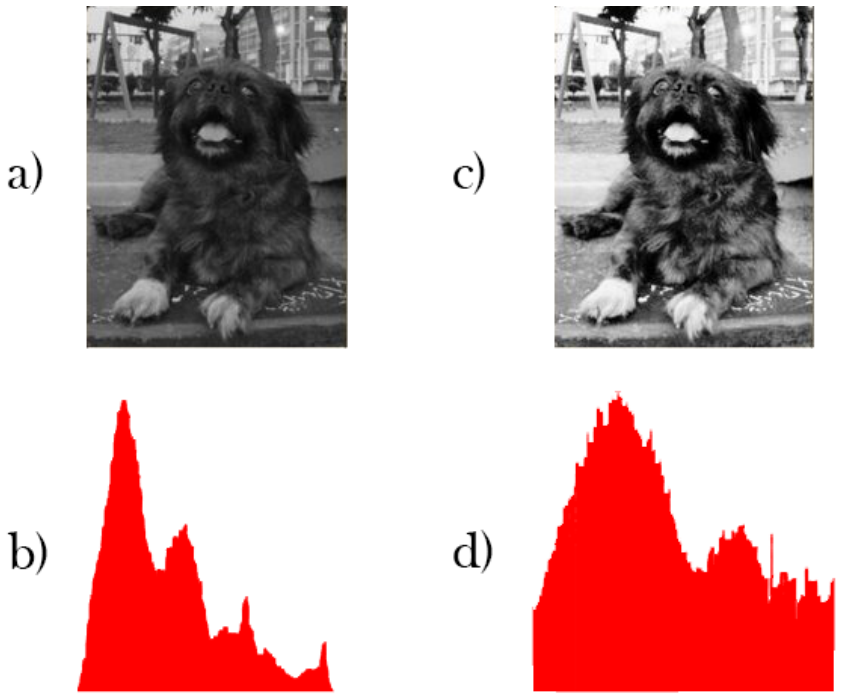
$$c(i) = \sum_{j=0}^{i} p(x_j). \tag{5}$$

Nyní chceme aby, transformace pixelu y byla ve svém oboru hodnot linearizována pomocí této definice:

$$y_i = c(i) \tag{6}$$

Nyní víme že c mapuje hodnoty do intervalu $[0, 1]$ a abychom dostali jejich původní hodnoty musíme aplikovat tuto transformaci:

$$y_i^{'} = y_i * (max - min) + min \tag{7}$$



Obrázek 20: Vyrovnání histogramu. a) původní obrázek, b) nevyrovnaný histogram, c) obrázek po vyrovnání histogramu, d) vyrovnaný histogram[3]

3.6 Filtry

Filtry ve zpracování obrazu slouží většinou k odstranění nežádoucích jevů, nejčastěji se jedná o šum. K tomu, abychom mohli šum z obrazu odstranit, se neobejdeme bez znalosti charakteristiky daného šumu. Výběr vhodného filtru tedy záleží na typu šumu. Šum se může objevit v obraze jako náhodný, který může být způsoben vadnými CCD elementy kdy se tento šum nazývá „sůl a pepř“. Dále se můžeme potkat s Gaussovským šumem, kdy každý pixel v obraze je mírně

pozměněn. Níže jsou popsány dva druhy filtrů: Gaussův filtr a bilaterální filtr. Oba filtry redukují šum, ale bilaterální filtr, na rozdíl od Gaussova filtru, zachovává hrany. Proto tyto filtry musíme volit s ohledem na jejich aplikaci.

Ve zpracování obrazu se filtry často aplikují s využitím konvoluce [1]. Diskrétní konvoluce se provádí pomocí konvoluční masky, kdy v masce jsou spočítány hodnoty na základě vybrané funkce. Střed masky přiložíme na příslušné místo v obraze, kde následně každý pixel překrytý maskou vynásobíme hodnotou v příslušné buňce a na závěr provedeme jejich součet. Tímto postupem dostaneme novou hodnotu pixelu a tu uložíme do nového obrazu. Konvoluční masky mohou mít různé velikosti. Konvoluce je tedy dána vzorcem:

$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x - i, y - j) * h(i, j). \tag{8}$$

3.6.1 Gaussův filtr

Gaussův filtr slouží pro odstranění šumu z obrazu [11]. Pro jeho implementaci je nejjednodušší způsob použití s konvoluční maskou. Konvoluční maska 10 je vypočtena na základě Gaussovy funkce 9 kde σ značí standardní odchylku Gaussovy distribuce, x a y vzdálenost od počátku na vertikální a horizontální ose.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \tag{9}$$

Tato maska se poté vynásobí s každým pixelem a všemi jeho sousedy v obraze. Masky mohou mít různé velikosti 3x3 nebo 5x5. Výsledná hodnota se uloží na dané místo v novém obraze. Tímto dosáhneme rozmazaného obrázku, kdy neuvidíme tolik šumu, ale zároveň budeme mít rozmazané i hrany, což může být problém při následné detekci hran.

$$\frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 1 & 16 & 26 & 16 & 4 \\ 4 & 26 & 41 & 26 & 7 \\ 7 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{10}$$

3.6.2 Bilaterální filtr

Bilaterální filtr redukuje šum v obraze přičemž zachovává hrany [11]. Hodnota každého pixelu v obraze je nahrazena váženým průměrem hodnot okolních pixelů, kdy vážený průměr může být založen na Gaussově rozdělení. Důležitým faktorem je, že vážený průměr nezávisí pouze na euklidovské vzdálenosti jako je tomu u Gaussova filtru, ale závisí také na radiometrický rozdílech (např. rozsahu barevné intenzity). Díky tomu jsou zachovány hrany a vážený průměr je pro každý pixel jiný. Zpravidla jsou důležité tři parametry d, σ_r s σ_d. Parametr d značí velikost

okolí od počítaného pixelu pro filtrování. Čím se parametr σ_r zvyšuje tím se rozdílnější barvy vedle sebe mísí dohromady, a tak vznikají větší jednobarevné plochy. Čím více σ_d roste tím více se vzdálenější pixely stejné barvy ovlivňují. Výpočet pixelu bilaterální filtrem je dán vzorcem:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p - q||) G_{\sigma_r}(|I_p - I_q|) I_q, \tag{11}$$

kde p je pixel obrazu pro který je hodnota počítána, q je pixel v oblasti S - konvoluční masky. I_p a I_q označují intenzity pixelů p a q. $G_{\sigma_s}(||p - q||)$ označuje váhu pixelu jako funkci vzdálenosti od středového pixelu. $G_{\sigma_r}(|I_p - I_q|)$ označuje váhu pixelu od středového pixelu jako rozdíl intenzit jasů těchto dvou pixelů.

Obrázek 21: Vlevo: původní obraz, uprostřed: obraz po aplikaci Gaussova filtru, vpravo: obraz po aplikaci bilaterálního filtru

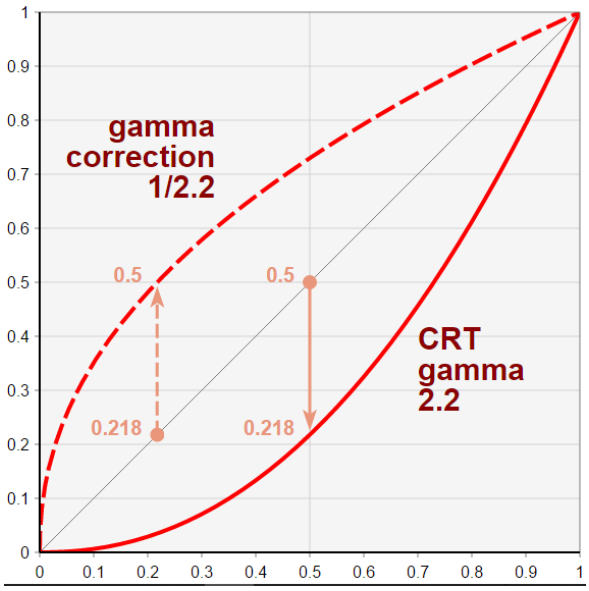
3.7 Gamma korekce

Gamma korekce upravuje rozložení barev v celém rozsahu co nejefektivněji a to úzce souvisí s lidským zrakem. Lidské vnímání intenzity světla je při slabém osvětlení intenzivnější, než při silném osvětlení, abychom během slunečného dne neoslepli. Díky této vlastnosti se člověk dokáže zorientovat i v tmavé místnosti.

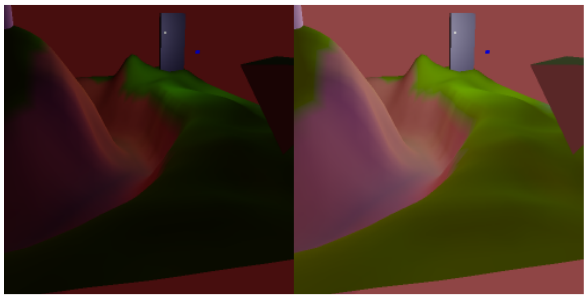
Problém nastává při pořízení obrazu, protože tehdy vzniká při dopadu daného množství světla na optický snímač signál, který se následně převádí na obraz. To ovšem znamená, že při dopadu dvojnásobku fotonů na snímač vznikne dvojnásobně větší signál. Tedy při snímání fotoaparátem nebo videokamerou je průběh funkce viz Obr. 22 mezi množstvím světla a výsledným signálem lineární, což je v rozporu s lidským zrakem. Gamma korekce nám tedy umožní lépe využít bitovou hloubku obrazu a rozložit barvy tak, jak je vnímáme my lidé. V praxi to znamená, že stínům, které jsou seskupeny na užším prostoru, dáme větší prostor viz Obr. 23 (zesvětlíme).

Gamma korekce je také obsažena v knihovně OpenCV a je dána jednoduchým vztahem 12. Kdy se může koeficient γ zvolit např. 1/2.2 dle křivky na Obr. 22.

$$I_{out} = I_{in}^{\gamma} \tag{12}$$



Obrázek 22: Průběh funkce gamma korekce[24]



Obrázek 23: Ukázka gamma korekce. Vlevo: původní obraz, vpravo: po gamma korekci[24]

3.8 Cannyho detektor hran

Detekci hran lze provádět pomocí konvoluce s různými maskami. John F. Canny v roce 1986 popsal optimální detektor[25], který se zaměřil zejména na následující tři kritéria:

- **Nízká míra chyb** - detekce pouze existujících hran
- **Přesná lokalizace** - hrana existující ve skutečnosti se musí nacházet na co nejpřesnějším místě v obraze
- **Minimum duplikací** - pokud možno každá hrana by měla být detekována jednou

Postup detekce:

1. Odfiltrujeme šum. Nejčastěji se používá k tomuto účelu Gaussův filtr s velikostí masky 5x5, ale můžeme použít i bilaterální filtr.
2. Nalezneme gradienty intenzit jasů v obraze pro směr G_x a G_y. K výpočtu gradientů můžeme použít Sobelovu masku, kdy pro osu G_y ji transponujeme. Poté z nalezených gradientů určíme jejich velikost G a směr Θ. Směr poté zaokrouhlíme do jednoho ze 4 možných směrů $(0°, 45°, 90° a 130°)$.

3. Následuje metoda ztenčení, která nám zajistí, že hrana bude detekována jen v místě jejího největšího gradientu. Při aplikaci to znamená, že pixel, který má být hranou, musí mít po obou stranách pixely s nižší intenzitou.

4. Prahování probíhá dvojí. Volíme dolní a horní práh. Canny ve svém článku doporučuje použít prahy horní:dolní v poměru mezi 2:1 a 3:1.

   - Pokud je gradient pixelu větší než horní práh, pak je zvolen za hranu
   - Pokud je gradient pixelu nižší než dolní práh, pak je odmítnut za hranu
   - Pokud je gradient pixelu mezi oběma prahy a zároveň jeho libovolný soused je větší než horní práh, poté je zvolen také za hranu

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{13}$$

$$G = \sqrt{G_x^2 + G_y^2}, \Theta = arctan\frac{G_y}{G_x} \tag{14}$$

3.9 Houghova transformace

Houghova transformace slouží k detekci jednoduchých tvarů v obraze na základě parametrického popisu hledaného objektu v obraze. Umožňuje nám hledat přímky, elipsy a kružnice. Kružnice je možné parametricky zapsat ve vztahu:

$$r^2 = (x - x_0) + (y - y_0), \tag{15}$$

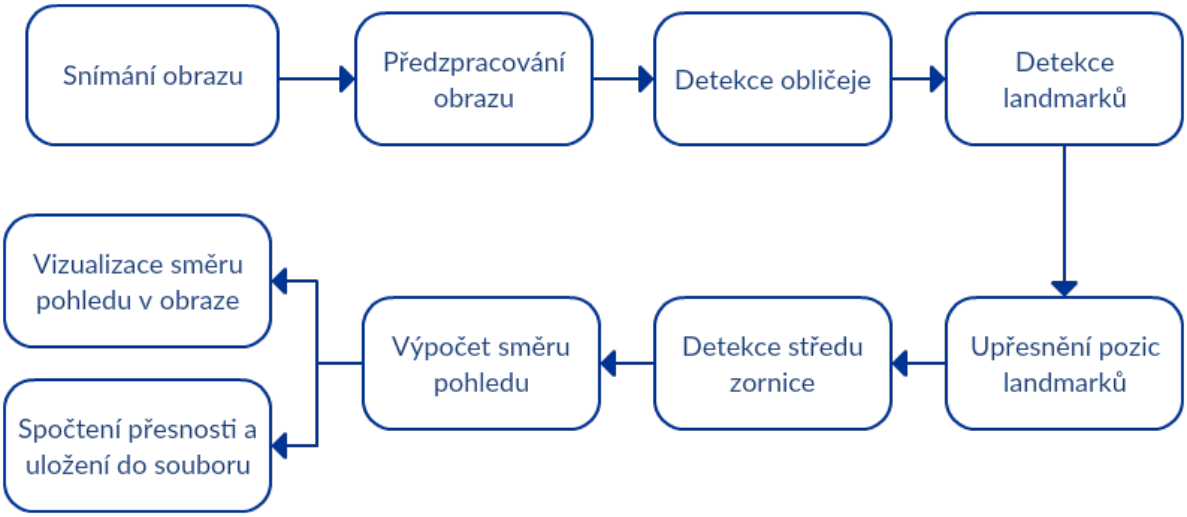
kdy r je poloměr kružnice, x a y souřadnice bodů v obraze a x_0, y_0 jsou souřadnice středu kružnice. Postup detekce probíhá následovně. Vytvoříme si trojrozměrný akumulátor pro středy a poloměry kružnic do kterých se budou inkrementovat středy kružnic. Pro každý bod, který je na kružnici, měníme x_0, y_0 a dopočítáváme r. Z toho vyplývá, že jakýkoliv nehranový body je možným středem kružnice. Pokud se v akumulátoru objevují nějaké středy x_0 a y_0 často, tak to znamená že se pravděpodobně jedná o kružnici se středem x_0, y_0 a poloměrem r. Výsledkem jsou tedy souřadnice středu nalezených kružnic a jejich poloměry.

4 Implementace

Tato kapitola se zabývá popisem samotné implementace detektoru směru pohledu. Kapitola obsahuje postup, kterým je získáván vstupní obraz, jak se upravuje a zpracovává. Následně jsou popsány metody detekce landmarků obličeje a detekce středu zornice. Pro detekci landmarků je popsán i způsob, pomocí kterého jsou pozice detekovaných landmarků očí upraveny blíže k očním víčkům. Na základě předem popsaných algoritmů probíhá výpočet směru pohledu. K ukázce jsou vyobrazeny příklady, kdy detekce landmarků nebo středu zornice nebyla úspěšná. Pro dosažení vyšší přesnosti byly použity buffery, které eliminují nechtěné rychlé přeskoky mezi detekovanými částmi ve snímcích jdoucích po sobě. Dále je obsažen popis a ukázka implementace OpenMP direktiv, které umožňují zpracování výpočtů ve více vláknech. Některé metody, a to zejména ty pro trénování detekce landmarků, jsou totiž časově náročné. V závěru této kapitoly je popsáno ovládání programu.

4.1 Řetězec zpracování obrazu pro detekci směru pohledu

K dosažení cíle, zhotovení detektoru směru pohledu, je potřeba provést několik důležitých kroků, které zobrazuje Obr. 24.



Obrázek 24: Jednotlivé kroky algoritmu detekce směru pohledu

Prvním krokem je pořízení videozáznamu a jeho načtení do programu. V druhém kroku je nutné obraz v každém snímku upravit pomocí předzpracování obrazu. To znamená převést jej do jednobarevného obrazu, normalizovat tento obraz, aplikovat vybrané filtry, provést gamma korekci a vyrovnání histogramu. Tyto operace s obrazem byly blíže popsány v teoretické části této práce. Dalšími kroky jsou detekce obličeje, která se provádí pomocí HAAR klasifikátoru a knihovny OpenCV, detekce landmarků a detekce středu zornice. Po detekci středu zornice, spočtení velikosti očí a jejich středů jsme schopni v dalším kroku spočíst směr pohledu. V posledním

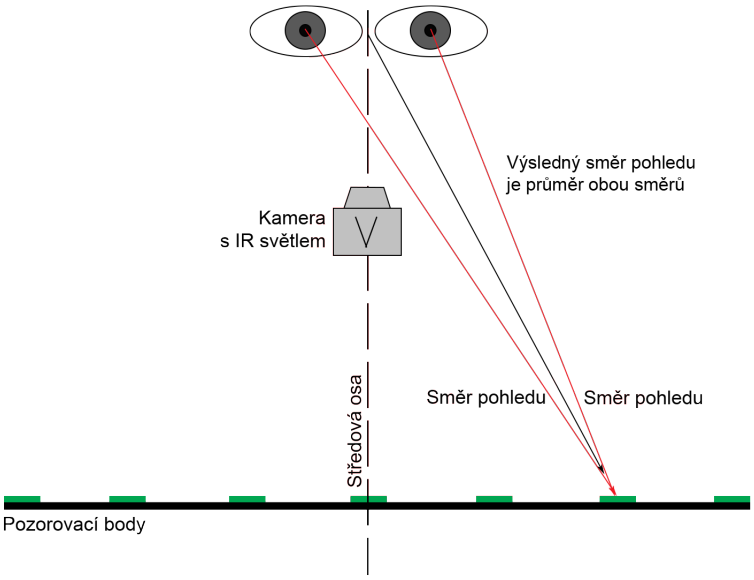
kroku detekce směru pohledu vizualizujeme do obrazu šipku reprezentující směr pohledu. Míru vychýlení středu zornice od středu oka, tedy směr pohledu v pixelech, je možné uložit do výstupního textového log souboru.

4.1.1 Soustava pro pořízení obrazu

K pořízení obrazu, ve kterém detekujeme směr pohledu, byla použita kamera od firmy μEye, která je citlivá na IR osvětlení viz Obr. 25. Software pro záznam videa byl použit rovněž od společnosti μEye. Dále bylo použito IR světlo k osvětlení pozorované osoby obsahující 48 IR diod. Pořizování videozáznamu probíhalo v laboratoři, kdy kamera byla umístěna na stativu, nad kterým bylo umístěno IR osvětlení.

Obrázek 25: Vlevo: použité IR osvětlení, vpravo: kamera značky μEye citlivá na IR osvětelní

Tato soustava byla umístěna před uživatelem, který zaměřoval svůj pohled na předem vyznačené body naproti němu viz Obr. 26. Kamera s IR světlem snímala obličej pozorované osoby a byla umístěna na středové ose ve výši očí. Celkem byly použity tři řady bodů pro vertikální pohyb oka, přičemž každá řada obsahovala 7 bodů pro horizontální pohyb oka. Bylo tedy možné zaznamenat směr pohledu od úhlu $-30°$ až do $30°$ horizontálně a $-10°$ až $10°$ vertikálně. Změna směru z jednoho bodu na druhý znamenala, změnu směru pohledu daným směrem o $10°$. Tyto videozáznamy byly použity i pro určení přesnosti detektoru. Na základě nahraných videozáznamů a předem známém směru pohledu bylo možné spočíst vychýlení středu zornice od skutečného středu oka, za účelem získání ground truth hodnot každého snímku.

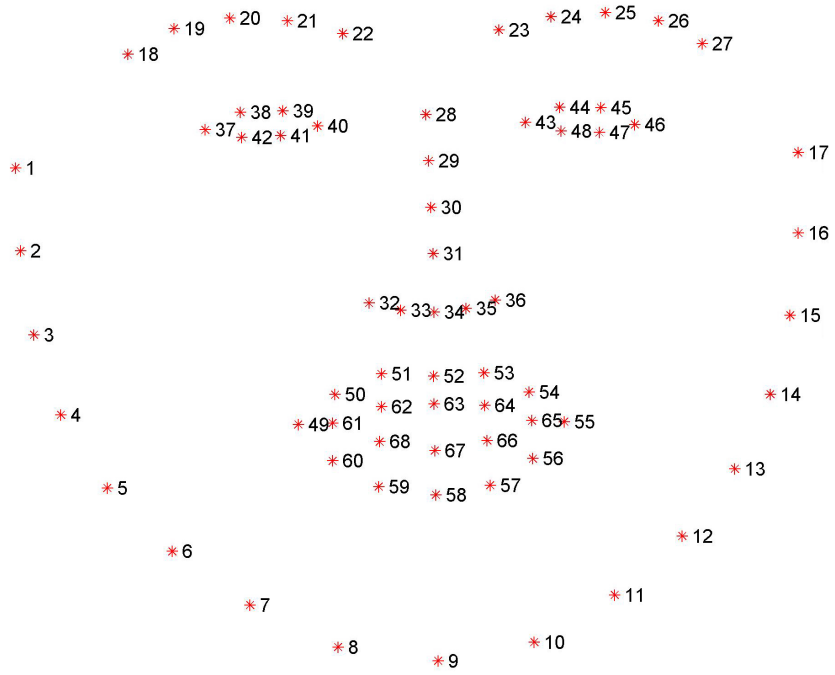


Obrázek 26: Schéma umístění kamery

4.2 Detekce landmarků a hranic oka

Tato podkapitola se zabývá detekcí významných částí obličeje tzv. landmarků. Landmark je jeden význačný bod v obraze, který leží na hranici, např. obličeje, obočí, oka nebo úst. Z více landmarků můžeme získat hranice jednotlivých částí obličeje v obraze viz Obr. 27. Po získání co nejpřesnějších okrajů očí jsme schopni určit skutečný střed oka, který je důležitý pro následující metodu určení směru pohledu.

K získání co nejpřesnějších hranic oka byl použit detektor založený na článku *Face Alignment at 3000 FPS via Regressing Local Binary Features* [26]. Procesy trénování rozpoznávání landmarků a jejich detekce byly implementovány v open-source projektu *Face alignment in 3000fps* [27], ze kterého je v této práci vycházeno.



Obrázek 27: 68 landmarků pro detekci obličeje [28]

4.2.1 Princip metody učení detekce landmarků

Článek [26] se zabývá postupným odhadováním (regresí) význačných bodů obličeje. Metoda je založena na učení landmarků obličeje v obraze, kdy se nezávisle pro každý landmark kóduje lokální binární charakteristika okolí vybraného landmarku. Jako spolehlivý přístup pro přesnou a robustní detekci obličejových landmarků, bylo využito minimalizace chyby mezi skutečným landmarkem a detekovaným landmarkem. Tento přístup totiž předpovídá tvar obličeje a jeho částí S na základě kaskádového postupu. Na začátku kaskády je počáteční tvar S^0, kdy S je postupně upravován odhadováním tvaru ΔS stupeň po stupni. V každém následujícím stupni, ve kterém se daný landmark vyhledává, se jeho okolí postupně zmenšuje. Obecně tvar ΔS^t v kroku t je odhadnut následovně:

$$\Delta S^t \;=\; W^t \Phi^t(I, S^{t-1}), \tag{16}$$

kdy I je vstupní obraz, S^{t-1} je tvar z předešlého kroku, Φ^t je mapovací funkce landmarků a W^t je lineární projekce landmarků na obličej. Φ^t a W^t jsou na počátku neznámé. Algoritmus tedy v dalším kroku přidává informaci o předešlém tvaru obličeje ΔS^t do ΔS^{t-1}.

V této metodě je učení rozděleno do dvou částí. V první části se učí mapovací funkce landmarků na obličej. Mapovací funkce Φ^t je rozdělena do lokálních mapovacích funkcí Φi^t, která reprezentuje vždy konkrétní landmark. Každá Φi^t se učí lokální binární charakteristiku okolí daného landmarku. Následným spojením všech lokálních binárních charakteristik, dostaneme Φ^t reprezentující celkový odhadnutý tvar obličeje. Jako učící funkce Φi^t pro lokální binární charak-

teristiky okolí landmarku je využito algoritmu random forest. Binární charakteristiky okolí tedy indikují, zda daná část obsahuje hledané vzory, či nikoliv.

Následně probíhá učení lineární projekce W^t, což reprezentuje rozmístění jednotlivých landmarků vůči sobě v obraze. Tento učící proces je také opakován v každém stupni kaskády.

4.2.2 Datasety a trénování

Pro natrénování detekce landmarků bylo potřeba získat datasety. V úvahu připadaly dvě možnosti, a to vytvořit si vlastní dataset, jehož tvorba by byla zdlouhavá a jeho výsledky by nemusely být dostačující, nebo získat dataset z volně dostupných zdrojů. Při hledání datasetů byly nalezeny vhodné datasety [28], které jsou určeny pro učení rozpoznávání landmarků obličeje. V těchto datasetech se nachází obličeje v různých stupních zakrytí, pod rozdílnou intenzitou osvětlení, v odlišném prostředí a od různých osob. Data jsou tvořena obrazy typu PNG nebo JPG, ve kterých se nachází obličej a soubor ke každému obrázku s 68 definovanými body obličeje viz Obr. 27. Nejprve je však nutné vytvořit soubor s cestami k jednotlivým obrázkům datasetu, který se poté volá v metodě *TrainModel*. Soubor slouží jako seznam obrázků určeného datasetu, které se mají načíst v procesu trénování. Soubor s cestami lze vytvořit pomocí příkazového řádku ve vybraném adresáři pomocí $dir\ /b/s/p/w\ *.jpg > Path\_Images.txt$. Po natrénování se vytvoří soubor *LBF.model*, který je následně použit pro detekci landmarků.

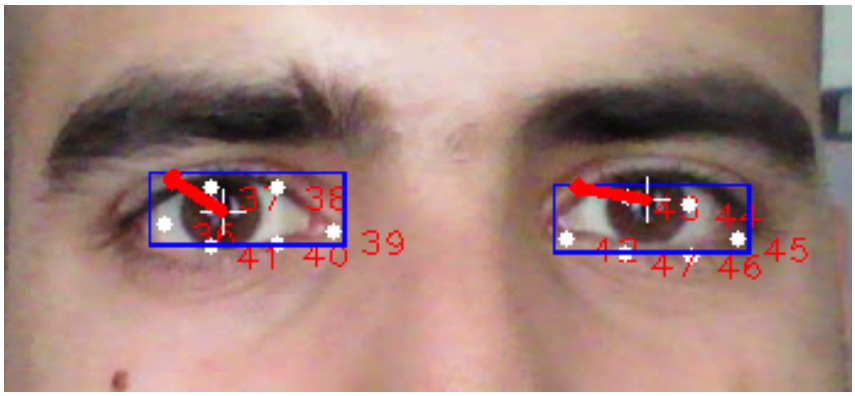
Trénování probíhalo na šesti různých datasetech o celkové velikosti 6360 vzorů. Celková doba trénování těchto vzorů byla okolo pěti hodin při využití 4 jader procesoru. Přesnost natrénovaného modelu je okolo 96,7% se správností určeného landmarku. Celý proces trénování byl urychlen pomocí direktiv OpenMP pro paralelní zpracování.

4.2.3 Detekce landmarků v kódu

V metodě *FaceDetectionAndAlignment* probíhá detekce obličeje a landmarků. Nejdříve bylo nutné načíst klasifikátor pro detekci obličeje (v našem případě *haarcascade_frontalface_alt.xml*) umístěný v adresáři *classifiers*. Při hledání obličeje pomocí klasifikátoru používáme předzpracovaný obraz, který je předzpracován vyrovnáním histogramu a převeden do formátu *CV_BGR2GRAY*. V obraze nás zajímá pouze jeden obličej, a to ten, který je největší, proto byl zvolen příznak klasifikátoru *CV_HAAR_FIND_BIGGEST_OBJECT*. Důvodem je, že by v obraze mohl být další obličej (v praxi při použití v automobilu by se mohlo jednat o spolujezdce sedícího na zadním sedadle). Výsledkem je výřez nalezeného obličeje, který je potřeba pro detekci landmarků. Detekce obličeje klasifikátorem trvá přibližně 15ms, záleží však na velikosti vstupního obrazu.

Pomocí metody *Predict* dojde k nalezení landmarků obličeje z výřezu obrazu detekovaného klasifikátorem a nalezené landmarky jsou uloženy do vektoru bodů. V našem případě si vystačíme pouze s body, které ohraničují okolí očí, protože ostatní landmarky jsou pro naši implementaci zbytečné. To znamená, že pro reprezentaci okolí očí, nám postačí landmarky $36 - 41$ pro pravé

oko a $42 - 47$ pro levé oko viz Obr. 27. Z těchto bodů jsme schopni spočíst rozměry oka a jeho střed. Vzhledem k tomu, že landmarky nejsou vždy detekovány přesně viz. Obr 28, je potřeba poopravit jejich umístění.



Obrázek 28: Chybně detekované landmarky

V případě, kdy landmarky nebyly posunuty do krajů očí, docházelo k nepřesnému výpočtu směru pohledu. Tato chyba byla viditelná již při pohledu před sebe, a to na bod s úhlem pohledu 0° vertikálně i horizontálně. Landmarky byly totiž detekovány uvnitř oka, nikoliv na jeho okraji.

4.2.4 Upřesnění landmarků

Upřesnění pozic landmarků bylo provedeno pomocí detekce hran očních víček s využitím Cannyho detektoru hran. Metoda byla použita na základě využití hran očních víček. Metoda pro upřesnění landmarků ohraňujících oko *correctionLandmarks* přijímá jako parametry tyto hodnoty: matici s obrazem pro vizualizaci detekce směru pohledu, matici s předzpracovaným obrazem a landmarky pro dané oko. Metoda vrací data obsahující 4 body reprezentující hranice oka, spočtené rozměry oka a spočtený střed oka. K detekci očních víček byl použit obraz vybraný dle detekovaných landmarků a zároveň byla zvětšena oblast zájmu (ROI) vertikálně i horizontálně. Důvodem jejího zvětšení bylo, že detekovaná oční víčka mohou být až za landmarky, čímž by následně nedošlo k nalezení jejich hran. Míru zvětšení okolí kolem landmarků pro osu x i y je možno upravit v konfiguračním souboru *config.txt*, kde můžeme upravit konstanty *kExpandOffsetX* a *kExpandOffsetY* v pixelech. Pro reprezentaci hranic oka byly ponechány pouze 4 landmarky, protože pro horní a dolní okraj oka nepotřebujeme dva landmarky, ale vystačíme si pouze s jedním, a to tím, se kterým je velikost oka větší.

Před detekcí hran je možno v konfiguračním souboru povolit aplikaci bilaterálního filtru, který v obraze zredukuje šum a přitom nerozmaže hrany. V konfiguračním souboru se jedná o atribut *enableBilateralFilterEyeImage* a dále můžeme upravit i hodnoty pro rozmazání *kBilateralFilterSigmaColor* a *kBilateralFilterSigmaSpace*.

Detekce hran pomocí metody *Canny* vyžaduje horní a dolní práh. Velikost Sobelovy masky je ponechána na velikosti 3x3. Prahy je možné upravit v konfiguračním souboru povolením vlast-

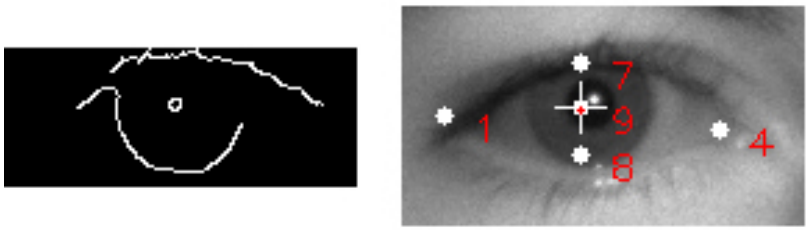
ních prahů atributem *enableYourCanny*, následně zadáním dolního prahu *cannyLowThreshold* a *cannyHighThreshold*. V případě, kdy není povoleno prahování v konfiguračním souboru, jsou prahy počítány automaticky. Prahy jsou zvoleny na základě výpočtu střední hodnoty z histogramu viz. výpis kódu 1. Poměr horního prahu a dolního prahu je doporučeno volit od 2:1 až do poměru 3:1. Z tohoto důvodu byla střední hodnota histogramu vynásobená pro dolní práh konstantou 0,66 a pro horní práh konstantou 1,33.

```
if (data.getenableYourCanny()){
   Canny(cannyImg, cannyImg, data.getcannyLowThreshold(), data.
      getcannyHighThreshold(), 3);
}else{
   Canny(cannyImg, cannyImg, 0.66*medianHist(notEq), 1.33*medianHist(notEq), 3)
      ;
}
...
double medianHist(cv::Mat mat)
{
   double cnt = (mat.rows*mat.cols) / 2;
   int bin = 0;
   double med = -1.0;

   int histSize = 256;
   float range[] = { 0, 256 };
   const float* histRange = { range };
   cv::Mat hist;
   cv::calcHist(&mat, 1, 0, cv::Mat(), hist, 1, &histSize, &histRange, true,
      false);

   for (int i = 0; i < histSize && med < 0.0; ++i)
   {
      bin += cvRound(hist.at< float >(i));
      if (bin > cnt && med < 0.0)
         med = i;
   }
   return med;
}
```

Výpis 1: Detekce hran očních víček a výpočet střední hodnoty histogramu

Po detekci hran bylo nezbytné identifikovat hranice očních víček ze vzniklého binárního obrazu. Pro identifikaci hran byla použita metoda *findContours* z knihovny OpenCV. Metoda

přijímá binární obraz, vektor hran složený z jednotlivých bodů, do kterého jsou uloženy hrany a vektor pro uložení hiearchie jednotlivých hran. Následně pro každý směr oka (nahoru, dolů, vlevo a vpravo) bylo provedeno prohledání hran bod po bodu a byl uložen vždy ten bod, který byl nejblíže k hledané straně pro danou hranu. Po prohledání krajních bodů hran, bylo nalezeno minimum v každém směru, a pokud bylo toto minimum větší než původní landmark, pak jej tento bod nahradil. Tato metoda velmi zvýšila přesnost detekce směru pohledu. V některých případech však nastaly problémy s detekcí spodního víčka, jelikož to je někdy nedostatečně výrazné viz. Obr. 29. V těchto případech se tedy nepodařilo identifikovat dolní okraj víčka a původní landmark se posunul maximálně na dolní okraj duhovky. K eliminaci tohoto jevu byl v konfiguračním souboru vytvořen atribut *bottomLandmarkOffsetY*, který posune spodní landmark o daný počet pixelů níže viz tabulka 3.



Obrázek 29: Vlevo: obraz po detekci hran s chybějící hranou spodního víčka. Vpravo: špatně detekovatelné spodní víčko

4.3 Detekce středu zornice

Pro úspěšné zjištění směru pohledu je potřeba nalézt v obraze zornici a určit její střed. Jako vstupní obraz byl použit výřez oka, který byl definován pomocí detekce a upřesnění landmarku. K detekci středu zornice lze využít různých metod a tyto metody lze roztřídit do několika kategorií.

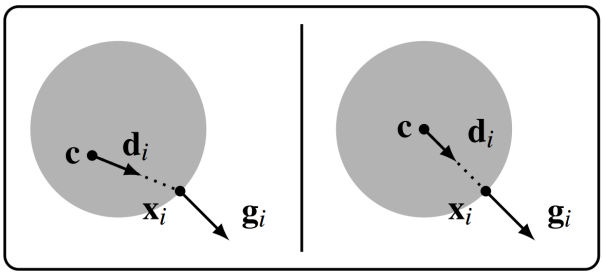
Dělení metod detekce středu zornice:

1. metody vycházející z vlastností oka (např. detekce podle barvy nebo hran),
2. metody založené na porovnání s již předem známými objekty,
3. kombinované metody vycházející z využití metod založených na vlastnostech a modelech.

4.3.1 Detekce zornice pomocí gradientů

Samotná detekce zornice je založená na změně gradientů v obraze a patří do kategorie metod vycházejících z vlastností oka. Metoda byla implementována dle článku[29] vydaným Fabianem Timmem a Erhardtem Barthem na univerzitě v Lübecku. Tato metoda má hned několik výhod, mezi které se řadí zejména vyšší robustnost pro detekci zornice než u ostatních metod k

tomu určených, nebo schopnost detekovat střed zornice i z obrazu pořízeného v nízkém rozlišení (webkamera). Další výhodou je, že metoda je invariantní vůči změně velikosti oka, pozici středu zornice, zhoršení kontrastu a různým intenzitám osvětlení.
Metoda se zaměřuje na analýzu vektorového pole obrazových gradientů. Autoři odvodili matematickou formulaci vlastností vektorového pole pro výpočet středu oka. Střed oka je definován jako střed kružnice nebo polokružnice, ve kterém se protíná nejvíce gradientů viz Obr.30.



Obrázek 30: Umělý příklad s tmavým kruhem na bílém pozadí simulující duhovku a oční bělmo. Vlevo: posunutý vektor **d** vůči vektoru **g**, které nemají stejnou orientaci. Vpravo: směr obou vektorů je shodný [29]

Pokud c je předpokládaný střed oka, x_i je pozice v obraze a g_i je směr růstu, pak normalizované posunutí vektoru d_i má stejnou orientaci jako vektor gradientu (g_i). S využitím vektorového pole gradientů, lze toto pole využít k výpočtu skalárního součinu mezi normalizovaným posunutí vektorů d_i a gradientu vektorů g_i. Vektorové pole gradientů se musí spočíst pro osu x a y. Vektorové pole bylo spočteno s využitím Sobelova operátoru. Viz výpis kódu 2 pro výpočet vektorového pole. Vektorové pole pro osu y je spočteno stejnou metodou jako pro osu x s tím rozdílem, že obraz je předem transponován.

```
cv::Mat computeMatXGradient(const Mat &src){
cv::Mat dst(src.rows, src.cols, CV_64F);

for (int y = 0; y < src.rows; ++y){
   const uchar *SRCr = src.ptr<uchar>(y);
   double *DSTr = dst.ptr<double>(y);

   DSTr[0] = SRCr[1] - SRCr[0];
   for (int x = 1; x < src.cols - 1; ++x){
      DSTr[x] = (SRCr[x + 1] - SRCr[x - 1]) / 2.0;
   }
   DSTr[src.cols - 1] = SRCr[src.cols - 1] - SRCr[src.cols - 2];
}

return dst;
```

```
}
```

Výpis 2: Výpočet vektorového pole

Optimální střed $c*$ kruhového objektu v obraze na pozicích všech pixelů x_i, $i \in \{1, ..., N\}$ je dán vztahem 17

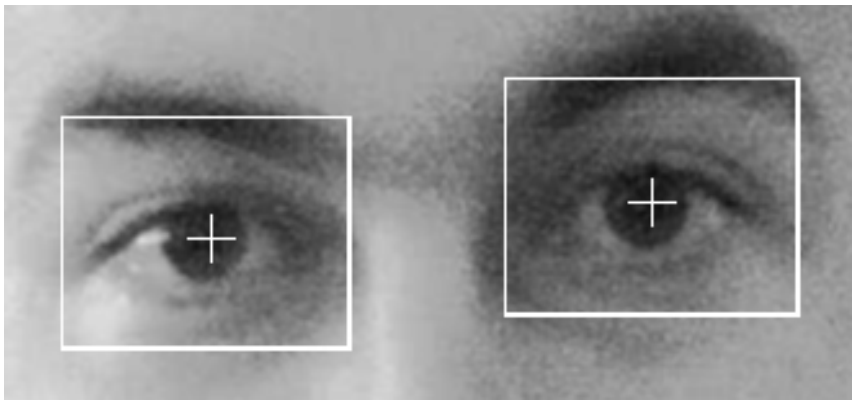
$$c^* = \underset{c}{\operatorname{argmax}} \left\{ \frac{1}{N} \sum_{i=1}^{N} (d_i^T g_i)^2 \right\} \tag{17}$$

,

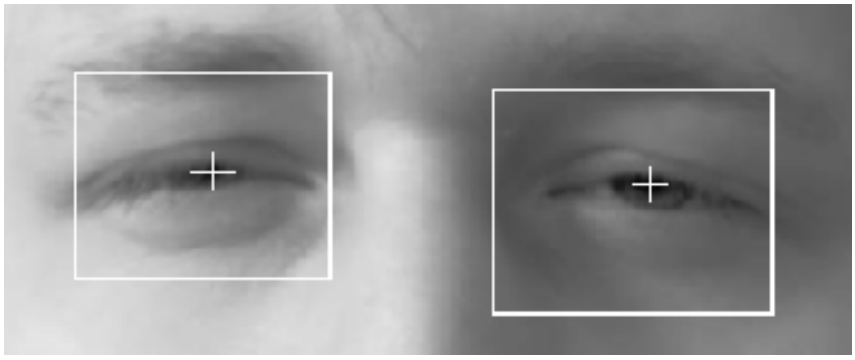
$$d_i = \frac{x_i - c}{||x_i - c||_2}, \forall i : ||g_i||_2 = 1 \tag{18}$$

Vektory posunutí d_i jsou zmenšeny do jednotkové délky, díky čemuž se zvýšila robustnost vůči lineárním změnám v osvětlení a kontrastu. Z důvodu složitosti tohoto algoritmu, která je $O(N^4)$, bylo provedeno zmenšení výřezu oka na šířku 80 pixelů, kdy výška se zmenšuje v poměru k šířce. Šířku oka pro detekci zornice je možné upravit v konfiguračním souboru pod parametrem *kFastEyeWidth*. K této složitosti dochází, protože zjištění nejpravděpodobnějšího středu zornice se pro každý pixel v obraze počítá jako suma gradientů procházejících potencionálním středem zornice. Výpočetní složitost dále klesla v případě, kdy byly ignorovány vektory gradientů v homogenních částech obrazu, kde nebyla výrazná změna směru.

Odchylka od skutečného středu zornice a toho, jenž byl detekován touto metodou, byla v průměru čtyři pixely. Tato metoda se osvědčila jako velmi robustní i při zhoršeném osvětlení a kontrastu. Střed zornice byl dobře detekovatelný i v případě, kdy oči v obraze byly přivřené viz Obr. 32.



Obrázek 31: Detekce zornice při zhoršeném osvětlení a kontrastu



Obrázek 32: Detekce zornice při extrémní přivření očí

Problém však nastal v případě, kdy oko bylo částečně překryto stínem nebo pokud měl uživatel příliš výrazné horní řasy a kontrast s očním bělmem byl velmi nízký (zejména při pohledu do strany). V těchto případech docházelo ke špatné detekci zornice, která byla detekována v řasách nebo v kraji očního víčka. K této chybě dochází, jelikož je v obraze detekováno více lokálních extrémů. Pro minimalizaci této chyby slouží konstanta w_c, která je spočtena pro každý potencionální střed oka. Střed oka bude správně nalezen s větší pravděpodobností, pokud je intenzita v bodě (c_x, c_y) invertovaného a vyfiltrovaného obrazu pomocí Gaussova filtru vyšší.

Výsledná rovnice pro výpočet středu zornice po přidání parametru w_c článku [29]:

$$c^* = \underset{c}{\operatorname{argmax}} \left\{ \frac{1}{N} \sum_{i=1}^{N} w_c (d_i^T g_i)^2 \right\}, w_c = I(c_x, c_y) \tag{19}$$

Ve zdrojovém kódu slouží pro detekci středu zornice metoda *findEyeCenter* přijímající vstupní obraz s obdélníkem definujícím oko a dále obraz, do kterého je vykreslován nalezený střed zornice. Obraz v obdélníku definujícím oko je prvně vyfiltrován Gaussovým filtrem, kdy následně v konfiguračním souboru můžeme zapnout odstranění odlesku IR přísvitu. Metoda *scaleToFastSize* je určena ke zmenšení obrazu obsahujícího oko na uživatelem definovanou šířku pro rychlejší výpočet středu zornice. Následně jsou spočteny gradienty pro osu x a y, kdy pro osu y je obraz transponován a spočten stejnou metodou jako pro osu x viz výpis kódu 2. Velikost gradientů je spočtena pomocí metody *matrixMagnitude*. Následně je provedena normalizace gradientů, jejich prahování a spočtení invertované matice vstupního obrazu pro parametr w_c. Ve výpisu kódu 3 je zobrazen výpočet sumy gradientů protínajících každý pixel vstupního obrazu a počet průniků gradientů pro každý pixel je uložen v matici *outSum*. V místě matice *outSum*, kde je největší počet průniků, je obsažen střed hledané zornice.

```
#pragma omp parallel for
for (int y = 0; y < weight.rows; ++y){
   const double *Xr = gradientX.ptr<double>(y);
   const double *Yr = gradientY.ptr<double>(y);
   for (int x = 0; x < weight.cols; ++x){
```

```
            double gX = Xr[x];
            double gY = Yr[x];
            if (gX == 0 && gY == 0){
               continue;
            }
            testPossibleCentersFormula(x, y, weight, gX, gY, outSum);
         }
      }
```

Výpis 3: Spočtení sumy gradientů protínajících každý pixel obrazu

4.3.2 Detekce zornice pomocí kružnic

Tato metoda je založena na principu detekce kružnice využitím Houghovy transformace, kterou tvoří oční duhovka a zornice. Tato metoda je značně jednodušší ve srovnání s předešlou metodou. Vstupním obrazem pro tuto metodu může být binární obraz nebo libovolný obraz, ve kterém jsou obsaženy hrany. Tyto hrany však nemusí být zcela zřetelné. Výhodou této metody je její robustnost vůči nepravidelnostem a přerušením hledané kružnice. Nevýhodou je, pokud kružnice není ve vstupním obraze dostatečně intenzivní.

K této implementaci byla využita metoda *HoughCircle* obsažená v knihovně OpenCV. Tato funkce hledá v obraze kružnice a příjmá tyto parametry:

- vstupní obraz, ve kterém se detekuje kružnice,
- výstupní pole vektorů, do kterého jsou uloženy informace o nalezených kružnicích (střed x, y a poloměr r),
- horní a dolní práh pro detekci hran (Canny hranový detektor),
- minimální a maximální poloměr detekované kružnice.

Pro hledání kružnic byly použity stejné prahy jako pro detektor hran v předešlé metodě. Minimální poloměr zornice byl volen o velikosti 5 pixelů a maximální o velikosti 30 pixelů. Po nalezení kružnic bylo nutné vybrat tu, která reprezentovala zornici. Bohužel nastaly případy, kdy zornice nebyla detekována, a nebylo tudíž možno určit střed zornice. Kvůli tomuto zásadnímu problému nebyla tato metoda nakonec použita v detektoru směru pohledu.

4.4 Výpočet směru pohledu

Spočtení směru pohledu probíhá v metodě *directionalVector* obsažené uvnitř souboru *LandmarksUtils* viz výpis kódu 4. Směr pohledu je průměrován na základě výpočtu směru pro obě oči a počítán v každém snímku po předchozích úpravách. Použitím průměrování se zamezilo tomu, že pozorovaná osoba v případě špatně detekované jakékoliv části (nepřesnému středu zornice nebo

nepřesným landmarkům) "nešilhala". Vstupy pro výpočet jsou dva body reprezentující skutečný střed očí a dva body reprezentující střed detekované zornice. Spočtený směr pohledu je vrácen jako vektor reprezentující směr pohledu pozorované osoby.

```
Vec3f directionalVector(Point detectLeftCenter, Point realLeftCenter, Point
    detectRightCenter, Point realRightCenter){
   float lX = detectLeftCenter.x - realLeftCenter.x;
   float lY = detectLeftCenter.y - realLeftCenter.y;

   float rX = detectRightCenter.x - realRightCenter.x;
   float rY = detectRightCenter.y - realRightCenter.y;

   float rSize = sqrt(rX * rX + rY *rY);
   float lSize = sqrt(lX * lX + lY *lY);

   Vec3f vec = Vec3f((lX + rX) / 2, (lY + rY) / 2, (rSize + lSize) / 2);
   return vec;
}
```

Výpis 4: Spočtení směru pohledu

Při implementaci však došlo k dalšímu problému a to hned po spočtení směru pohledu. Jelikož i nadále během detekce docházelo k mírný posunům detekovaných částí nutných pro výpočet směru, musely být tyto posuny eliminovány. K tomu bylo použito bufferů.

4.5 Buffery

Jelikož detekce směru pohledu nebyla uspokojující, zejména kvůli rychlým přeskokům výsledného směru pohledu, byly implementovány buffery pro redukci tohoto jevu. Přeskoky byly způsobeny nepřesnou detekcí landmarků a středu zornice v daném snímku. Z toho vyplývá, že když směr pohledu zůstal stejný, ale detekce byla odlišná oproti předchozímu snímku, došlo k přeskoku. Buffer slouží jako fronta, do které se ukládají data předchozích snímků. Pro detektor směru pohledu bylo vhodné implementovat dva buffery. Jeden buffer *landmarksBuffer* byl vytvořen pro ukládání jednotlivých landmarků, detekovaného středu zornice a vypočteného středu oka. Druhý buffer byl vytvořen pro ukládání směru a velikosti výsledného směru pohledu *directionBuffer*.

Při každém novém spočtení a upřesnění landmarku, byl tento landmark spolu se spočteným středem oka a detekovaným středem zornice vložen do *landmarksBuffer* nakonec fronty, přičemž první z nich byl odstraněn. Tato fronta se vynásobila s předem vytvořenými prioritami pro každý prvek v bufferu. Velikost bufferu je stanovena na poslední 4 snímky a podle velikosti bufferu jsou stanoveny priority (0,1;0,15;0,25;0,5), kdy nejnižší priorita je vždy pro nejstarší snímek.

Spočtení výsledného směru pohledu spolu s jeho velikostí se provádí uvnitř bufferu *directionBuffer*. V bufferu se opět vynásobí poslední uložené směry pohledu se stejnými prioritami jako

v případě *landmarksBuffer*. Velikost tohoto bufferu je stanovena tak, aby byla schopna pojmout informace o posledních čtyřech snímcích. Použitím bufferů bylo docíleno eliminace rychlých přeskoků mezi jednotlivými snímky.

4.6 Urychlení pomocí OpenMP

Algoritmy pro trénování detekce landmarků a detekce středu zornice spotřebovaly velké množství výpočetního času. Proto bylo nutné jejich zrychlení, k němuž bylo použito direktiv OpenMP. OpenMP (Open Multi-Processing) je aplikační programovací rozhraní API, které umožňuje více vláknové programování se sdílenou pamětí v programovacích jazycích C, C++ a Fortran napříč různými operačními systémy. OpenMP se skládá ze souboru direktiv určených pro překladač, knihovních postupů a proměnných ovlivňujících běh programu.

Princip OpenMP spočívá v rozdělení části kódů do více vláken. To znamená, že část kódu, která je určena k paralelnímu zpracování, je označena direktivou kompilátoru. Tato direktiva způsobí vytvoření podprocesů (vláken) před spuštěním této části. Každé vlákno má přiřazeno celé identifikační číslo (ID), kdy hlavní vlákno má ID=0. Hlavní výhodou je, že každé vlákno provádí svou část nezávisle na ostatních vláknech. Po provedení paralelního kódu jsou následně tyto vlákna spojeny do hlavního vlákna, které pokračuje dále.

OpenMP je součástí programovacího prostředí Visual Studio, kdy stačí ve vlastnostech projektu povolit použití této knihovny a zadat počet vláken pro zpracovávání. Následně pro každou část kódu, kde bylo potřeba rozdělit hlavní vlákno, bylo použito knihovních direktiv. Ukázka rozdělení kódu s direktivou *#pragma omp parallel for*, která provede rozdělení následující smyčky do vláken viz 5.

```
#pragma omp parallel for
for (int j = 0; j<num_residual; j++){
   tmp = predict(models[j], binfeatures[i]);
   if (j < num_residual / 2){
      deltashape_bar(j, 0) = tmp;
   }
   else{
      deltashape_bar(j - num_residual / 2, 1) = tmp;
   }
}
```

Výpis 5: Ukázka použití direktivy OpenMP

4.7 Ovládání programu

Tato podkapitola popisuje ovládání programu detektoru směru pohledu. Program se spouští pomocí příkazového řádku, kdy se následně zobrazí okno s videem a průběhem detekce směru

pohledu.

Pro spuštění programu musí být v adresáři se spustitelným souborem konfigurační soubor pojmenovaný *config.txt*. Konfigurační soubor obsahuje hodnoty, kterými lze upravovat běh programu před spuštěním bez nutnosti měnit hodnoty ve zdrojové kódu, které by se následně musely kompilovat. V souboru musí být obsaženy všechny atributy, přičemž na jejich uspořádání nezáleží. Na každém řádku musí být uveden jeden atribut a k němu za znakem '=' uvedena hodnota. Výchozí nastavení konfiguračního souboru může vypadat dle tabulky 2 v příloze, ve které je popsána funkčnost každého atributu.

Po ověření, že je konfigurační soubor vložen, můžeme provést spuštění detektoru směru pohledu. V adresáři *video/groundTruth* se nacházejí 3 testovací videa. Pro videa *video*1.*avi* a *video*2.*avi* jsou vytvořeny i textové soubory. Tyto textové soubory obsahují referenční hodnoty, ve kterých je číslo snímku a střed detekované zornice posunutý vůči středu oka v pixelech. Program obsahuje 4 možné způsoby spuštění, z nichž každý provádí vždy jiné operace. Můžeme si spustit ukázku detekce směru pohledu na nahraném testovacím videozáznamu, spustit ukázku na videozáznamu z webkamery, natrénovat vlastní model pro detekci landmarků nebo otestovat přesnost vlastního modelu. Natrénování a otestování vlastního modelu se bohužel neobejde bez zásahu do zdrojového kódu.

- Pro spuštění ukázky z existujícího videozáznamu stačí spustit program *EyeGazeDetection* s parametry *Demo nazev_videa.avi*, kdy video musí být v adresáři *video*.

- Pro spuštění ukázky s videozáznamem pořízeného prostřednictvím webkamery je nutné spustit program *EyeGazeDetection* pouze s parametrem *Demo*. Po spuštění programu je zobrazeno okno, ve kterém je video pořízené webkamerou a detekovaný směr pohledu uživatele.

- Dalšími možnostmi je natrénování vlastního modelu pro detekci landmarků nebo kontrola přesnosti natrénovaného modelu (*TrainModel* nebo *TestModel*). K tomu jsou potřeba datasety, které je možné stáhnout ze zdroje [28] nebo použít z přiloženého DVD. U těchto datasetů je potřeba upravit cesty k souborům *Path_Images.txt* v každém adresáři s daným datasetem. Následně je nutné upravit načítání datasetu ve zdrojovém kódu.

Při pokusu o spuštění detektoru bez zadaných nebo chybně zadaných parametrů se zobrazí výpis možných parametrů pro spuštění.

5 Zhodnocení výsledků a přesnost

V této části jsou shrnuty výsledky měření přesnosti algoritmu určeného k detekci směru pohledu. Přesnost byla otestována na dvou videozáznamech, kdy na každém byla přítomná jiná osoba. V každém videozáznamu se výsledky přesnosti mírně liší, a to i v případě, že byl použit stejný konfigurační soubor. Oba videozáznamy jsou na přiloženém DVD k této práci.

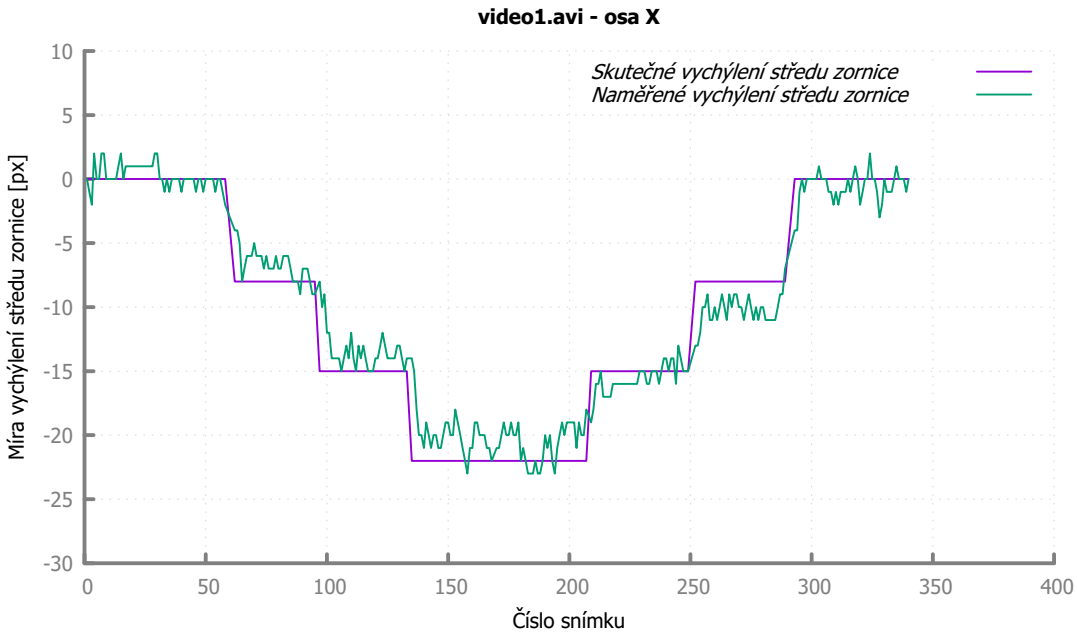
Přesnost je reprezentována jako rozdíl naměřeného vychýlení středu zornice ze středu oka a předem definovaného vychýlení pro daný snímek videozáznamu. Věděli jsme tedy na jaké místo, pozorovaná osoba zaměřuje svůj pohled. Mohli jsme tedy určit, o kolik pixelů byl vychýlen střed zornice ze středu oka v daném snímku pro obě osy. Stejným způsobem jsme mohli dále pokračovat pro další snímky a videozáznamy. Z těchto dat byly vytvořeny textové soubory, které obsahovaly číslo snímku a míru vychýlení pro osu X a Y. Při vytvoření těchto hodnot byly odstraněny snímky, na kterých docházelo k mrknutí. Takto jsme si vytvořili tzv. "ground truth" hodnoty. Změna směru pohledu o 10° podle osy X, znamenala změnu směru pohledu daným směrem v obou testovacích videích o 7,5 px. V ose Y, pak změna směru pohledu o 10° znamenala posun o 5 px. To znamená, že po naměření hodnot pomocí implementovaného detektoru, se musí hodnoty v pixelech vynásobit pro osu X konstantou *1,33* a pro osu Y konstantou 2.

Pro spočtení přesnosti směru pohledu je nutné mít vytvořený soubor s ground truth hodnotami. V konfiguračním souboru musí být povolena možnost *enableCheckPrecision* a zapsán název souboru s ground truth hodnotami v parametru *precisionFileName* viz tabulka 2 v příloze. Následně je nutné spustit program s vybraným videozáznamem. Po spočtení přesnosti je vyhodnocena celková přesnost v pixelech, kterou lze najít v nově vytvořeném textovém souboru s předponou *output*, nacházející se ve stejném adresáři jako soubor s ground truth hodnotami. Zároveň je do tohoto souboru zapsána očekávaná hodnota pro každý snímek a naměřená hodnota včetně jejich rozdílů. Pokud nejsou v souboru s ground truth hodnotami uvedena data pro některé snímky, pak přesnost pro tyto snímky není uvedena ani ve výstupním souboru.

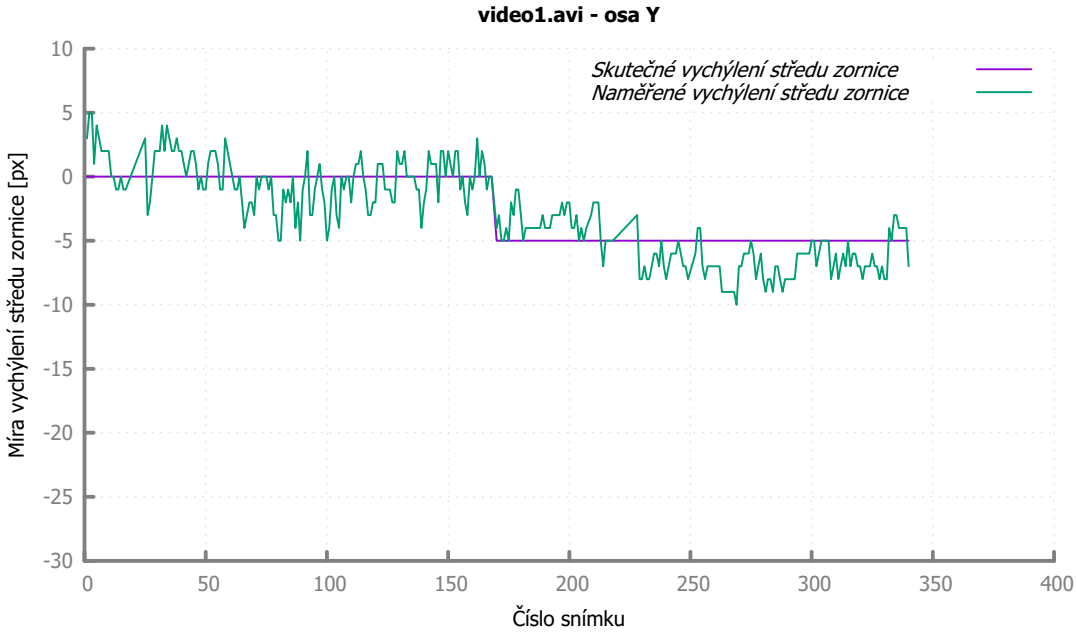
5.1 Testovací videozáznam - video1

V prvním případě byl testován videozáznam s názvem *video1.avi* viz Obr. 33. Posun spodního landmarku byl upraven v konfiguračním souboru na 5 pixelů. Grafy 34 a 35 znázorňují vypočtenou přesnost pro každý snímek v ose X a ose Y testovacího videozáznamu video1.avi.

Obrázek 33: Ukázka z testovacího videa - video1.avi. Obrázek vlevo znázorňuje pohled před sebe, obrázek uprostřed pohled vpravo, obrázek vpravo pohled nahoru.



Obrázek 34: Graf reprezentující přesnost detekce směru pohledu podle osy X v testovacím videozáznamu video1.avi

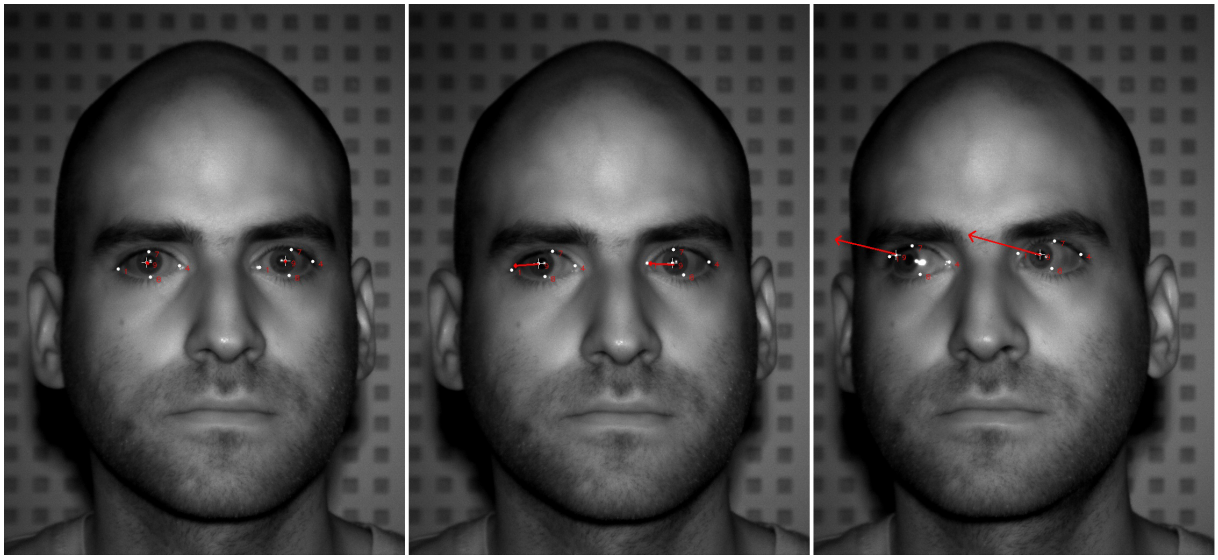


Obrázek 35: Graf reprezentující přesnost detekce směru pohledu podle osy Y v testovacím videozáznamu video1.avi

Přesnost videozáznamu video1.avi byla v průměru pro osu X 1,45 px $(1,93°)$ a pro osu Y 2,63 px $(5,26°)$. Větší nepřesnost pro osu Y byla způsobena tím, že míra vychýlení pro úhel 10° pro tuto osu, byla o 1/3 menší než pro osu X.

5.2 Testovací videozáznam - video2

Jako druhý byl testován videozáznam s názvem *video2.avi* viz Obr. 36. Posun spodního landmarku byl upraven v konfiguračním souboru na 10 pixelů. Grafy 37 a 38 znázorňují vypočtenou přesnost pro každý snímek v ose X a ose Y testovacího videozáznamu video2.avi.



Obrázek 36: Ukázka z testovacího videa - video2.avi. Obrázek vlevo znázorňuje pohled před sebe, obrázek uprostřed pohled vpravo, obrázek vpravo chybnou detekci středu zornice.

Obrázek 37: Graf reprezentující přesnost detekce směru pohledu podle osy X v testovacím videozáznamu video2.avi

Obrázek 38: Graf reprezentující přesnost detekce směru pohledu podle osy Y v testovacím videozáznamu video2.avi

Přesnost videozáznamu video2.avi byla v průměru pro osu X 2,71 px $(3,6°)$ a pro osu Y 1,92 px $(3,84°)$. Oproti testovacímu videozáznamu video1.avi byla zaznamenána větší průměrná chyba pro osu X, a to z důvodu, jelikož při pohledu do strany byl špatně detekován střed zornice. Střed zornice byl detekován v okraji oka, což zapříčinilo zvýšení průměrné nepřesnosti mezi snímkem 145 - 226 viz Obr. grafu 37.

5.3 Celková přesnost

Celková průměrná přesnost detektoru směru pohledu po spočtení průměrných přesností z obou videozáznamu je uvedena v tabulce 1.

	osa X	osa Y
Celková chyba (px)	2,08	2,275
Celková chyba (°)	2,77	4,55

Tabulka 1: Celková průměrná přesnost implementovaného detektoru směru pohledu

6 Závěr

Tato práce se zabývá problematikou detekce směru pohledu pomocí videokamery citlivé na IR světlo. V teoretické části je popsána problematika týkající se detekce směru pohledu a teorie použitých algoritmů v následující praktické části.

V praktické části je popsán řetězec zpracování obrazu pro detekci směru pohledu, včetně použité videokamery a IR světla. Metoda pro určení detekce směru pohledu je implementována na základě míry vychýlení středu zornice oka pozorované osoby od skutečného středu oka.

Pro nalezení středu oka byla použita metoda detekce landmarků obličeje. Obličej byl vyhledán kaskádovým HAAR klasifikátorem implementovaným v knihovně OpenCV. Algoritmus pro hledání landmarků byl převzat ze stávající implementace v projektu "Face Alignment in 3000 fps" [27]. Nejprve bylo nutné natrénovat algoritmus detekce landmarků. Toho bylo uskutečněno za pomocí již vytvořených datasetů, které obsahovaly pro každý snímek předem stanovené landmarky. Po detekci landmarků byl implementován algoritmus k upřesnění hranic oka. Jelikož landmarky nebyly vždy přesně detekovány, docházelo k chybnému spočtení středu oka a následnému chybnému výpočtu směru pohledu. Postup trénování detekce landmarků, včetně jejich následné detekce a jejich upřesnění je popsán v kapitole 4.2.

Pro určení středu zornice byl použit algoritmus založený na změně gradientů v obraze. Jako vstupní obraz pro tento algoritmus byl použit výřez obrazu na základě detekce landmarků. Tato metoda je popsána v kapitole 4.3 a vychází z tvrzení, že nejpravděpodobnější střed oka se nachází v místě, kde se protíná nejvíce gradientů. Z hlediska složitosti tohoto algoritmu byla implementována metoda pro zmenšení výřezu oka na stanovenou šířku dle konfiguračního souboru. Dále bylo využito direktiv OpenMP pro vícevláknové zpracování. Jako druhá metoda pro detekci středu zornice byla použita metoda založená na detekci kružnic. Jelikož tato metoda nebyla úspěšná, nebyla nakonec při implementaci detektoru směru pohledu použita.

Po získání hranic oka bylo možné spočíst jeho skutečný střed, který byl využit spolu s detekovaným středem zornice pro výpočet směru pohledu. Výsledný směr pohledu byl spočten, jako průměr směrů pohledů mezi oběma očima. Vizualizace detekovaného směru pohledu byla implementována, jako šipka směřující ze středu zornice na místo pohledu. Délka vizualizované šipky značí velikost úhlu směru pohledu. Čím více směřuje pohled pozorované osoby do strany, tím je šipka delší. Jelikož při vizualizaci docházelo ke kmitání šipky reprezentující směr pohledu, bylo použito bufferů, které tento jev eliminovaly.

Testování probíhalo na dvou videozáznamech, pro které byly vytvořeny ground truth hodnoty. V každém videozáznamu byla spočtena chyba, která vznikla mezi gound true hodnotami a naměřenými hodnotami. Celková chyba detektoru směru pohledu po zprůměrování vychází pro osu X $2,77°$ a pro osu Y $4,55°$. Testování je popsáno v kapitole 5 včetně výsledných grafů detekce směru pohledu.

V praktické části je dále popsáno ovládání programu pro detekci směru pohledu, včetně použití konfiguračního souboru. Jako možné vylepšení této implementace detektoru směru pohledu

je navrhováno využití další metody pro detekci středu zornice, a to za účelem její přesnější detekce. Metoda je nyní vhodná do místností s umělým osvícením a IR přísvitem. Intenzivní denní světlo by mohlo způsobit různé odlesky v okolí očí a zhoršit tímto celkovou přesnost detekce.

Literatura

[1] BRADSKI, Gary, et al. The opencv library. Doctor Dobbs Journal, 2000, 25.11: 120-126.

[2] JÓŹWIK, Agnieszka; SIEDLECKI, Damian; ZAJĄC, Marek. Analysis of Purkinje images as an effective method for estimation of intraocular lens implant location in the eyeball. Optik-International Journal for Light and Electron Optics, 2014, 125.20: 6021-6025.

[3] OpenCV. [online]. [cit. 2017-03-15]. Dostupné z: http://opencv.org

[4] Oko (a mozek) versus fotoaparát. [online]. [cit. 2017-03-15]. Dostupné z: http://www.fotoroman.cz/tech1/light_eye_camera.htm

[5] Způsoby sledování pohybu zraku. [online]. [cit. 2017-03-15]. Dostupné z: http://klimes.mysteria.cz/clanky/psychologie/ocnikamera_historie.pdf

[6] ROBINSON, David A. A method of measuring eye movemnent using a scieral search coil in a magnetic field. IEEE Transactions on bio-medical electronics, 1963, 10.4: 137-145.

[7] MOHAMED, Abdallahi Ould; DA SILVA, Matthieu Perreira; COURBOULAY, Vincent. A history of eye gaze tracking. 2007.

[8] Tobii Pro - Envision human behavior. [online]. [cit. 2017-03-15]. Dostupné z: http://www.tobiipro.com/

[9] Eye Snake - High Level Design. [online]. [cit. 2017-03-15]. Dostupné z: https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2005/dcc34/finalsite/design.html

[10] Elbert, T., Lutzenberger, W., Rockstroh, B., Birbaumer, N., 1985. Removal of ocular artifacts from the EEG. A biophysical approach to the EOG. Electroencephalogr Clin Neurophysiol 60, 455-463.

[11] PARIS, Sylvain, et al. Bilateral filtering: Theory and applications. Foundations and Trends® in Computer Graphics and Vision, 2009, 4.1: 1-73.

[12] Keren, A.S.; Yuval-Greenberg, S.; Deouell, L.Y. (2010). "Saccadic spike potentials in gamma-band EEG: Characterization, detection and suppression". NeuroImage. 49: 2248–2263. doi:10.1016/j.neuroimage.2009.10.057. PMID 19874901.

[13] Nielsen, Jakob. Pernice, Kara. (2010). "Eyetracking Web Usability."New Rideres Publishing. p. 11. ISBN 0-321-49836-4. Google Book Search. Retrieved on October 28, 2013.

[14] O'CONNELL, Stephen D., et al. Eye Tracking-Based Target Designation in Simulated Close Range Air Combat. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting. Sage Publications, 2012. p. 46-50.

[15] Joint Helmet Mounted Cueing System. [online]. [cit. 2017-03-15]. Dostupné z: http://www.best-of-flightgear.dk/jhmcs.htm

[16] FEJTOVÁ, M., et al. System I4Control®: Contactless control PC. In: Intelligent Engineering Systems, 2006. INES'06. Proceedings. International Conference on. IEEE, 2006. p. 297-302.

[17] Oční pohyby. [online]. [cit. 2017-03-15]. Dostupné z: http://www.ocni-pohyby.cz/handicapovani/

[18] MONTEIRO, Gonçalo; PEIXOTO, Paulo; NUNES, Urbano. Vision-based pedestrian detection using Haar-like features. Robotica, 2006, 24: 46-50.

[19] VIOLA, Paul; JONES, Michael. Rapid object detection using a boosted cascade of simple features. In: Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. IEEE, 2001. p. I-I.

[20] MARCOS, Ma Shiela Angeli C.; SORIANO, Maricor N.; SALOMA, Caesar A. Classification of coral reef images from underwater video using neural networks. Optics express, 2005, 13.22: 8766-8771.

[21] Local Representation of Facial Features. [online]. [cit. 2017-04-01]. Dostupné z: http://what-when-how.com/face-recognition/local-representation-of-facial-features-face-image-modeling-and-representation-face-recognition-part-1/

[22] DALAL, Navneet; TRIGGS, Bill. Histograms of oriented gradients for human detection. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005. p. 886-893.

[23] LBP, HOG. [online]. [cit. 2017-04-02]. Dostupné z: http://www.kky.zcu.cz/uploads/courses/mpv/05/materialy05.pdf

[24] MCKESSON, Jason L. Learning Modern 3D Graphics Programming. Arcsynthesis. org, 2012, 17.

[25] CANNY, John. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, 1986, 6: 679-698.

[26] REN, Shaoqing, et al. Face alignment at 3000 fps via regressing local binary features. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014. p. 1685-1692.

[27] C++ implementation of face alignment in 3000fps. [online]. [cit. 2017-04-08]. Dostupné z: https://github.com/yulequan/face-alignment-in-3000fps

[28] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, M. Pantic. 300 Faces in-the-Wild Challenge: The first facial landmark localization Challenge. Proceedings of IEEE Int'l Conf. on Computer Vision (ICCV-W), 300 Faces in-the-Wild Challenge (300-W). Sydney, Australia, December 2013.

[29] Timm, Fabian, and Erhardt Barth. "Accurate Eye Centre Localisation by Means of Gradients."VISAPP 11 (2011): 125-130.

A Obsah DVD

Na přiloženém DVD jsou následující soubory:

1. Zdrojové kódy k programu.
2. Spustitelný program EyeGazeDetection.
3. Text této diplomové práce ve formátu PDF.
4. Datasety k trénování detekce landmarků

B Konfigurační soubor

Název atributu	Výchozí hodnota	Datový typ atributu	Funkce atributu
kFastEyeWidth	80	int	Šířka obrazu výřezu oka pro rychlejší spočtení středu zornice.
kGradientThreshold	25.0	double	Práh pro detekci středu zornice.
kWeightBlurSize	5	int	Velikost masky pro filtr rozostření k detekci středu zornice.
kExpandOffsetX	15	int	Velikost zvětšení oblasti do každé strany v ose X pro upřesnění pozic landmarků.
kExpandOffsetY	10	int	Velikost zvětšení oblasti do každé strany v ose Y pro upřesnění pozic landmarků.
kBilateralFilterD	7	int	Velikost okolí pro filtrování bilaterálním filtrem.
kBilateralFilterSigmaColor	50	int	Čím více hodnota roste, tím více se rozdílnější barvy mísí a vznikají jednobarevné plochy.
kBilateralFilterSigmaSpace	50	int	Čím více hodnota roste, tím více se vzdálenější pixely ovlivňují.
kGamma	2.2	double	Míra gamma korekce.
enableDrawCross	true	bool	Vykreslení kříže detekovaného středu zornice do obrazu.
enableDrawLandmarks	true	bool	Vykreslení bodů (landmarků) ohraničujících oko do obrazu.
maxLengthArrow	5	int	Stupeň délky šipky znázorňující směr pohledu v obraze.
enableCheckPrecision	false	bool	Povolení načítání souboru pro spočtení přesnosti detekce směru pohledu.

Tabulka 2: Tabulka s atributy a parametry konfiguračního souboru 1. část

Název atributu	Výchozí hodnota	Datový typ atributu	Funkce atributu
precisionFileName	soubor.txt	string	Název souboru s referenčními daty pro výpočet směru pohledu. (Na řádku v souboru musí být uvedeno vždy číslo snímku, vychýlení středu zornice po ose X a po ose Y v pixelech. Tyto hodnoty musí být odděleny středníkem) Soubor se musí nacházet v adresáři *video/groundTrue*.
enableCannyEyeImage	true	bool	Zobrazení výřezu oka po detekci hran.
enableBilateralFilterEyeImage	true	bool	Povolení aplikace bilaterálního filtru pro upřesnění landmarků.
enableRemoveIR	false	bool	Povolení odstranění odlesků v zornici vzniklých IR přísvitem. (experimentální použití)
enableYourCanny	false	bool	Povolení vlastního prahování pro detekci očních víček k přesnějšímu umístění landmarků.
cannyLowThreshold	50	int	Hodnota dolní prahu pro detekci očních víček.
cannyHighThreshold	180	int	Hodnota horního prahu pro detekci očních víček.
enableLog	false	bool	Povolení ukládání vychýlení středu zornice vůči středu oka.
bottomLandmarkOffsetY	0	int	Hodnota o kolik se má automaticky posunout dolní landmark oka.

Tabulka 3: Tabulka s atributy a parametry konfiguračního souboru 2. část